POLITÉCNICO DE LEIRIA | ESCOLA SUPERIOR DE TECNOLOGIA E GESTÃO

Redes Neuronais Artificiais em *Deep Learning* na deteção da doença COVID-19

Licenciatura em Engenharia Informática

Pedro Marcelino Ferreira

Teresa Leal Ferreira

Leiria, setembro de 2022

POLITÉCNICO DE LEIRIA | ESCOLA SUPERIOR DE TECNOLOGIA E GESTÃO

Redes Neuronais Artificiais em *Deep Learning* na deteção da doença COVID-19

Licenciatura em Engenharia Informática

Pedro Marcelino Ferreira

Teresa Leal Ferreira

Trabalho de Projeto da unidade curricular de Projeto Informático realizado sob a orientação do Professor Doutor João da Silva Pereira

Leiria, setembro de 2022

Agradecimentos

Agradecemos ao nosso orientador Professor Doutor João da Silva Pereira, por estar sempre disponível para nos ajudar nas nossas dificuldades, pela paciência e simpatia, e por todo o conhecimento e sabedoria que nos transmitiu.

Agradecemos também aos nossos amigos e família pelo apoio que nos deram ao longo da realização deste trabalho.

Resumo

A doença COVID-19 é uma pneumonia e a sua infeção resulta numa inflamação e líquido nos pulmões. É uma doença cinco vezes mais mortal que a gripe tendo, por isso, uma elevada taxa de mortalidade. Pode ser, por vezes, confundida com uma gripe ou outras pneumonias virais pois é muito semelhante em radiografias do tórax, o que dificulta o seu diagnóstico.

Com o sucesso deste projeto será mais fácil e rápido diagnosticar a doença COVID-19 e consequentemente indicar um tratamento mais adequado a cada caso, o que poderá resultar na mitigação dos efeitos mais graves do vírus.

Neste contexto, a deteção da doença é feita através da análise das radiografias dos pulmões de vários pacientes através de redes neuronais artificiais em *Deep Learning*. Estas redes são treinadas e aperfeiçoadas de modo a ganharem cada vez mais conhecimento para situações futuras.

O objetivo deste trabalho é detetar esta doença, a COVID-19, através das radiografias e naturalmente obter uma elevada taxa de acerto nas imagens para que o nosso algoritmo seja credível e realista, de modo a ser possível utilizá-lo em contexto real.

Com a conclusão deste trabalho podemos observar que os nossos objetivos foram alcançados com sucesso, e apesar de não considerarmos que o produto final seja extremamente fiável para o fim para que foi concebido, é um bom auxílio nessa matéria e, portanto, estamos satisfeitos com o que conseguimos realizar e obter.

Palavras-chave: Redes neuronais artificiais, *Deep Learning*, *Efficient Net*, COVID-19

Abstract

COVID-19 disease it is a pneumonia and its infection results in inflammation and fluid in the lungs. It is a disease five times more deadly than the flu and therefore has a high mortality rate. It can sometimes be confused with flu or other viral pneumonia as it is very similar on chest X-rays, which makes it difficult to diagnose.

With the success of this project, it will be easier and faster to diagnose covid-19 disease and consequently indicate a more appropriate treatment for each case, which could result in the mitigation of the most serious effects of the virus.

In this context, the detection of the disease is done through the analysis of radiographs of the lungs of several patients through artificial neuronal networks in Deep Learning. These networks are trained and improved to gain more and more knowledge for future situations.

The objective of this work is to detect this disease, COVID-19, through radiographs and naturally obtain a high accuracy rate in the images so that our algorithm is credible and realistic, so that it can be used in a real context.

With the conclusion of this work, we can observe that our objectives have been achieved successfully, and although we do not consider that the final product is extremely reliable for the purpose for which it was designed, is a good help in this matter and, therefore, we are satisfied with what we have achieved and obtained.

Keywords: Artificial neuronal networks, Deep Learning, Efficient Net, COVID-19

Índice

Lista de Figuras

Lista de siglas e acrónimos

CNN	*Convolutional Neural Network* (Redes Neuronais Convolucionais)
ReLU	*Rectified Linear Unit*
FLOPS	*Floating point operations per second*
COVID	*Coronavirus Disease*
SARS-COV-2	*Severe Acute Respiratory Syndrome Coronavirus 2*
API	*Application Programming Interface*
AUC	*Area Under ROC Curve* (Área sob curva)
NP	*Negative for Pneumonia* (Negativo para a Pneumonia)
IA	*Indeterminate Appearance* (Aparência Indeterminada)
TA	*Typical Appearance* (Aparência Típica)
AA	*Atypical Appearance* (Aparência Atípica)

Introdução

O presente trabalho foca-se na deteção da doença COVID-19, *Coronavirus Disease* 2019, através de redes neuronais em *Deep Learning*. A doença COVID-19 é provocada pelo coronavírus SARS-COV-2, *severe acute respiratory syndrome coronavirus 2*, e é uma doença que afeta maioritariamente o sistema respiratório podendo, por vezes, ser confundida com uma simples gripe. Esta doença é cinco vezes mais mortal que a gripe causando assim uma mortalidade significativa. É uma pneumonia e a infeção pulmonar por COVID-19 resulta numa inflamação e líquido nos pulmões. O COVID-19 parece muito semelhante a outras pneumonias virais e bacterianas em radiografias de tórax, o que dificulta o diagnóstico.

Para a realização deste trabalho tivemos como base a competição "SIIM-FISABIO-RSNA COVID-19 Detection" que decorreu na plataforma *Kaggle*, esta consistia em desenvolver um algoritmo capaz de detetar a doença COVID-19 através da análise das radiografias. As imagens e ficheiros que utilizámos foram os dados para a realização da competição [22].

Se este modelo for bem-sucedido irá ajudar os radiologistas a diagnosticar a doença COVID-19, nos milhões de pacientes, com mais confiança e rapidez. Permitirá também que os médicos vejam como a doença se fez sentir nos pulmões de cada paciente e, assim, indiquem um tratamento mais adequado a cada estado. Como resultado de um diagnóstico mais rápido os pacientes receberão mais rapidamente os cuidados para a sua condição, o que poderá ajudar na mitigação dos efeitos mais graves do vírus [22].

A deteção da doença é feita através da análise de radiografias dos pulmões de vários pacientes, e para isso são criadas e treinadas redes neuronais artificiais em *Deep Learning* que identificam padrões comuns nas várias imagens sendo possível obter um resultado.

Uma rede neuronal é como um cérebro humano virtual, esta é composta por neurónios que são treinados permitindo assim que as aplicações identifiquem padrões e consequentemente resolvam problemas no ramo da Inteligência Artificial.

Este tema é atual e de elevada pertinência devido ao facto de a doença COVID-19 ainda estar presente no nosso dia a dia, não tanto como antes, mas ainda assim tem algum tipo de impacto. Com o desenvolvimento deste trabalho é possível detetar esta doença de uma forma

rápida e sem causar dor para o paciente, visto que apenas é necessária uma radiografia dos seus pulmões.

Outro motivo pelo qual escolhemos este tema foi porque em Portugal ainda não existia nenhuma aplicação que detetasse esta doença e então vimos este trabalho como uma mais-valia, tanto para nós porque pudemos desenvolver e aprofundar os conhecimentos em redes neuronais e *Deep Learning*, bem como para a sociedade porque é algo que os irá beneficiar, neste trabalho foi direcionado para a doença COVID-19, mas também pode ser aplicado para outras doenças.

O objetivo deste trabalho consiste em detetar a doença COVID-19 através da análise de radiografias de vários pacientes, esta deteção é feita através da criação e do treino de neurónios que constituem as redes neuronais artificiais em *Deep Learning*. Temos também como objetivo obter uma elevada percentagem de acerto nas radiografias, ou seja, fazer com que o treino das redes neuronais seja o mais preciso possível de modo a corresponder com a realidade.

Para treinar as redes neuronais, é utilizado um conjunto de imagens aleatórias que diferem em cada ciclo de modo a melhorar a aprendizagem das redes e proporcionar um melhor treino para, posteriormente, estas conseguirem obter um melhor desempenho em novas situações. Na avaliação do modelo final, são utilizadas as imagens que não foram usadas no treino e, assim, obtemos a percentagem de acerto deste modelo.

O trabalho apresenta a seguinte estrutura:

- O Capítulo 1 corresponde à introdução e descrição do tema
- O Capítulo 2 corresponde ao estado da arte
- O Capítulo 3 corresponde ao enquadramento teórico, este é composto por 3 secções:
    - A secção 3.1 corresponde a *Deep Learning*, esta ainda é composta por 2 subsecções:
        - A 3.1.1 corresponde a como funciona o *Deep Learning*
        - A 3.1.2 corresponde a aplicações do *Deep Learing*
    - A secção 3.2 corresponde a redes neuronais convolucionais
    - A secção 3.3 corresponde a *efficient net*, esta ainda é composta por 3 subsecções:
        - A 3.3.1 corresponde ao dimensionamento de modelos composto: uma maneira melhor de ampliar as CNNs
        - A 3.3.2 corresponde à arquitetura
        - A 3.3.3 corresponde ao desempenho
- O Capítulo 4 corresponde à descrição do processo, este é composto por 5 secções:

- A secção 4.1 corresponde à descrição do *Dataset* utilizado
- A secção 4.2 corresponde à descrição da primeira abordagem
- A secção 4.3 corresponde à descrição da segunda abordagem
- A secção 4.4 corresponde à descrição da terceira abordagem
- A secção 4.5 corresponde à descrição da quarta abordagem sendo esta a abordagem final

- O Capítulo 5 corresponde à descrição do processo, este é composto por 2 secções:
  - A secção 5.1 corresponde à introdução
  - A secção 5.2 corresponde à descrição da elaboração
  - A secção 5.3 corresponde à instalação e uso
- O Capítulo 6 corresponde à descrição do processo, este é composto por 4 secções:
  - A secção 6.1 corresponde à descrição da primeira abordagem
  - A secção 6.2 corresponde à descrição da segunda abordagem
  - A secção 6.3 corresponde à descrição da terceira abordagem
  - A secção 6.4 corresponde à descrição da quarta abordagem
- O Capítulo 7 corresponde à discussão dos resultados
- O Capítulo 8 corresponde às conclusões retiradas com a realização deste trabalho

Estado da arte

Os algoritmos de *Machine Learning* têm o potencial para estarem profundamente envolvidos em todos os campos da medicina, desde a descoberta de medicamentos até à tomada de decisões clínicas. O sucesso dos algoritmos de M*achine Learning* em tarefas de visão computacional nos últimos anos vem num momento oportuno em que os registos médicos são altamente digitais.

O uso de registos médicos eletrónicos (RME) quadruplicou de 11.8% para 39.6% entre médicos de consultório nos Estados Unidos desde 2007 até 2012. As imagens médicas são uma parte integrante dos RME e atualmente são analisadas por radiologistas humanos, que são limitados pela velocidade, fadiga e experiência. Leva anos e um grande custo financeiro para treinar um radiologista qualificado e alguns sistemas de saúde recorrem ao *outsourcing* para obter relatórios de radiologia, em países de baixo custo como a Índia via teleradiologia. Um diagnóstico tardio ou com falhas pode causar danos ao paciente, portanto é ideal que a análise de imagens médicas seja feita de forma automática, precisa e através de um algoritmo de *Machine Learning* eficiente.

As estruturas bidimensionais e tridimensionais de um órgão em estudo são cruciais para identificar o que é normal e o que é anormal. Ao manter essas relações espaciais locais, as CNNs são adequadas para realizar imagens tarefas de reconhecimento. As CNNs servem para diversos fins, incluindo classificação de imagem, localização, deteção, segmentação e registo. As CNNs são o algoritmo mais popular de *Machine Learning* em tarefas de reconhecimento de imagens e aprendizagem visual, devido à sua característica única de preservar as relações de imagem locais, enquanto executa redução da dimensionalidade. Desta forma, captura relacionamentos de características (*features*) importantes numa imagem (como por exemplo quando os pixéis de uma borda se unem para formar uma linha), e reduz o número de parâmetros que o algoritmo tem que processar, aumentando a eficiência computacional. As CNNs são capazes de receber tanto como entradas como também processar imagens bidimensionais, bem como imagens tridimensionais com pequenas modificações. Esta é uma vantagem útil na conceção de um sistema para uso hospitalar, pois tanto há a presença de raios X, que são bidimensionais, como também de tomografias computadorizadas ou ressonâncias magnéticas, que são volumes tridimensionais.

É importante salientar que as CNNs são exemplos de algoritmos de *machine learning* supervisionados que requerem quantidades significativas de dados de treino.

Neste caso estamos a tratar de um problema de classificação que, por vezes, também é chamada de deteção auxiliada por computador (*Computer-aided diagnosis – CADx*). Lo et al. descreveram uma CNN para detetar nódulos pulmonares em radiografias do tórax já em 1995 [1]. Foram usadas 55 radiografias de tórax e uma CNN com 2 camadas ocultas para dizer se uma região tinha ou não um nódulo pulmonar.

Rajkomar et al. [2] usaram *data augmentation* em 1850 imagens de radiografias ao tórax para obter 150 000 amostras de treino. Usando a CNN *GoogLeNet* [3] modificada e pré-treinada conseguiram classificar a orientação das imagens em vista frontal ou lateral com quase 100% de precisão. Apesar desta tarefa de classificação ter um uso limitado a nível clínico demonstra a eficácia do pré-treino e da *data augmentation* na aprendizagem de meta dados relevantes das imagens como parte de um diagnóstico eventualmente totalmente automático no fluxo de trabalho.

A pneumonia ou infeção no peito é um problema de saúde comum em todo o mundo que é eminentemente tratável, Rajpurkar et al. [4] usaram uma *DenseNet* [5] modificada com 121 camadas convolucionais chamada *CheXNet* para classificar 14 doenças diferentes vistas em raios-x do tórax, usando 112 000 imagens do *dataset ChestXray14*. O *CheXNet* alcançou desempenho de última geração na classificação das 14 doenças. A classificação de pneumonia, em particular, alcançou uma pontuação de Área Sob Curva (AUC) de 0,7632 com a análise *Receiver Operating Characteristics* (ROC). Além disso, num conjunto de testes de 420 imagens, o *CheXNet* igualou ou melhorou a performance de 4 radiologistas individuais e também a performance de um painel composto por 3 radiologistas.

Shen et al. [6] usaram CNNs combinadas com *Support Vector Machine* (SVM) e *Random Forest* (RF) para classificar nódulos pulmonares em benignos ou malignos com base em 1010 tomografias computadorizadas de pulmões do *dataset Lung Image Database Consortium* (LIDC-IDRI). Este método classificou os nódulos com 86% de precisão.

Esta informação foi retirada e pode ser consultada para mais pormenor aqui [7].

Ainda neste tópico Causey et al. apresentaram o *NoduleX*, uma abordagem sistemática para prever a malignidade do nódulo pulmonar a partir de dados de tomografias computorizadas, com base em redes neuronais convolucionais de *Deep Learning*. Para treinar e avaliar, analisaram mais de 1000 nódulos pulmonares em imagens do *Lung Image Database Consortium image collection* (LIDC-IDRI). Todos os nódulos foram identificados e classificados por quatro radiologistas torácicos experientes que participaram no projeto LIDC. O *NoduleX* conseguiu alcançar uma alta precisão para classificação da malignidade

do nódulo, com uma AUC de aproximadamente 0.99, que acaba por ir ao encontro da análise do conjunto de radiologistas experientes [8].

Bar et al. exploraram a possibilidade de uma CNN identificar diferentes tipos de patologias em imagens de raios-x. Além disso, como os *datasets* com muitos dados normalmente não estão disponíveis no domínio médico exploraram a viabilidade de usar uma abordagem de D*eep Learning* com base em aprendizagem não-médica. O algoritmo foi testado num *dataset* de 93 imagens. Usaram uma CNN que foi treinada com o *dataset ImageNet,* uma base de dados conhecida com imagens não médicas em grande escala. Conseguiram obter uma AUC de 0.93 para derrame pleural direito, 0.89 para deteção de coração aumentado e 0.79 para a classificação entre radiografia de tórax saudável e anormal, onde todas as patologias são combinadas numa grande classe. Esta foi uma experiência inédita que mostra que o *Deep learning* com *datasets* de imagens não médicas em grande escala pode ser suficiente para tarefas gerais de reconhecimento de imagens médicas [9].

Enquadramento Teórico

Deep Learning

O *Deep Learning* pode ser utilizado em muitas tarefas que envolvam a inteligência artificial minimizando assim a intervenção humana, um bom exemplo disso são os assistentes digitais, no telemóvel ou em aparelhos externos, que realizam comandos com base na deteção de voz.

Uma rede neuronal constituída por menos de três camadas é considerada uma rede neuronal básica (Figura1), se esta for constituída por mais de três camadas é considerada um algoritmo em *Deep Learning* (Figura 2), assim, podemos concluir que *Deep Learning* é uma rede neuronal composta por mais de três camadas. As redes neuronais são compostas por neurónios que simulam como o cérebro humano trabalha e os neurónios são treinados de modo a ficarem mais "inteligentes", ou seja, terem mais capacidade para acertarem no resultado. Quantas mais camadas uma rede neuronal possuir melhor pois vai ter mais neurónios para treinar e consequentemente mais ligações entre estes, tornando-os assim mais capacitados para prever situações futuras [10].

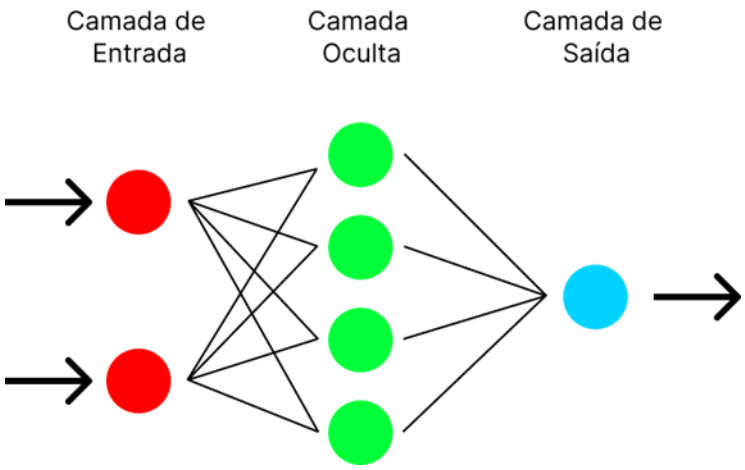

Figura 1 – Rede neuronal básica

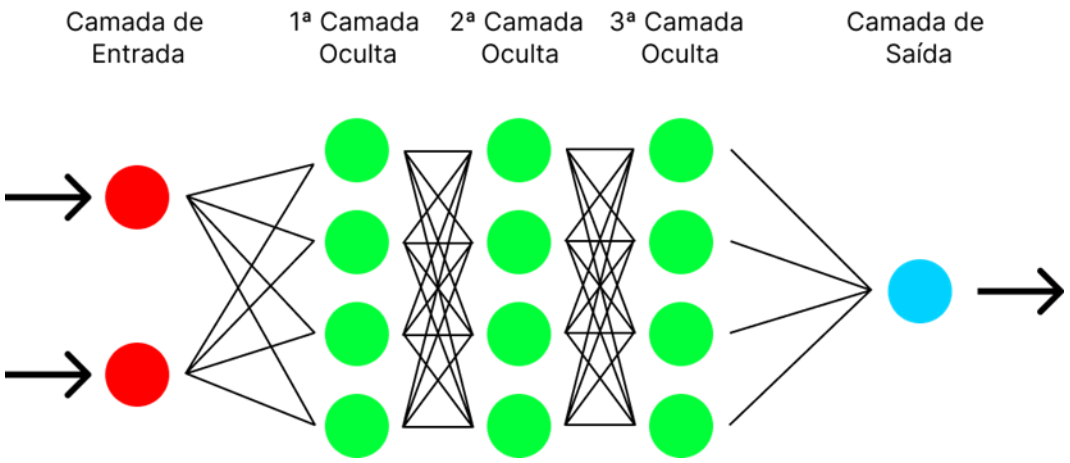

Figura 2 – Rede neuronal em Deep Learning

Este algoritmo é capaz de processar dados que podem ir desde texto a imagens, e consegue remover uma parte da intervenção humana. Podemos tomar como exemplo um conjunto de imagens de flores onde queremos organizar estas por categorias como "Tulipa". "Margarida", "Rosa", entre outras, o *Deep Learning* é capaz de escolher qual a característica importante para distinguir uma flor da outra.

Como funciona o Deep Learning

As redes neuronais em *Deep Learning* simulam como o cérebro humano funciona através de uma combinação de dados de entrada, pesos e *bias*. A junção destes três componentes faz com que seja possível determinar e classificar corretamente e com precisão os diferentes objetos dentro dos dados.

Como já referido anteriormente, uma rede neuronal é composta por várias camadas e estas, por sua vez, são constituídas por vários neurónios que estão ligados a vários ou a todos os neurónios da camada seguinte. Quando os dados são transmitidos entre os neurónios são aplicados pesos na sua entrada bem como o *bias*, ou seja, os dados quando vêm de um neurónio e entram noutro neurónio levam juntamente um peso e um *bias*. O peso diz respeito à força da ligação entre os dois neurónios, quanto maior for o seu valor mais importante será essa ligação que se reflete na saída. O *bias* é uma variável constante que pode assumir qualquer valor, mas normalmente assume o valor de 1, e diz respeito a uma entrada adicional, em cada neurónio, que é responsável por controlar o valor no qual a função de ativação é ativada [11].

A função de ativação consiste na transformação do valor de entrada do neurónio no valor de saída que, por sua vez, vai servir de entrada para o neurónio da camada seguinte. Esta

função é dada (figura 3) pela soma dos produtos do valor de entrada com o seu peso correspondente, somando no final o valor do *bias*, depois de calculado este valor aplicamo-lo na função de ativação resultando no valor de saída [13]. Existem vários tipos de funções de ativação e consoante o problema aplicasse a mais favorável para o sucesso do mesmo. No caso do nosso projeto utilizámos a função de ativação ReLU, *Rectified Linear Units*, esta função é bastante simples e consiste em devolver o valor de saída do neurónio se este for positivo, caso seja negativo devolve o valor 0 (zero). Podemos então escrever esta função da seguinte forma: *f(r)=max(0,r)*, ou seja, o máximo entre 0 (zero) e o valor de saída (r), [14].

As várias camadas de nós que constituem uma rede estão todas ligadas através destes, que, por sua vez, baseiam-se na camada anterior de modo a conseguirem melhorar a sua previsão. Podemos chamar a camada de entrada, que é onde o modelo recebe os dados, e a camada de saída, que é onde é saem os resultados de camadas visíveis, todas as outras são chamadas de camadas ocultas.

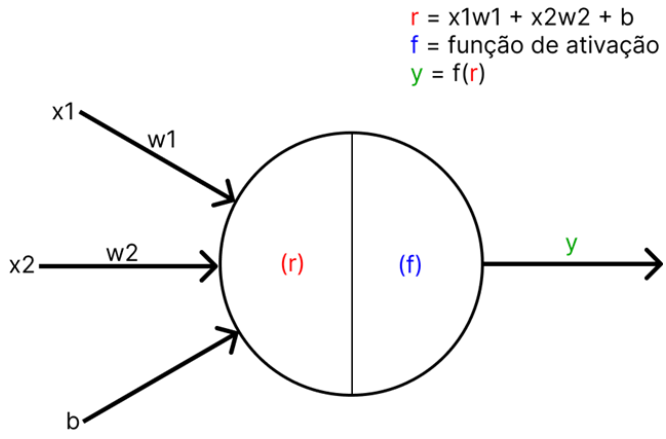

Figura 3 – Exemplo de como funciona um neurónio

Aplicações do *Deep Learning*

Todos os dias estamos em contacto com dispositivos que utilizam *Deep Learning* mas nem sempre nos apercebemos disso devido ao facto de este estar tão bem integrado no dispositivo. Podemos tomar como exemplo as seguintes situações:

- Aplicação da Lei: é possível detetar padrões em *Deep Learning* e, com isto, jogar a nosso favor, ou seja, utilizá-lo para detetar padrões que sejam considerados de ações criminosas ou menos corretas. Analisando gravações de vídeo e/ou voz, e aplicando o *Deep Learning*, é possível investigar um grande número de dados em pouco tempo, o que faz com que seja mais rápido identificar situações criminosas e assim agir mais rapidamente.

- Atendimento ao Cliente: hoje em dia quando contactamos um serviço que presta apoio ao cliente por via telefónica passamos por uma triagem, ou seja, o que está a falar connosco não é uma pessoa, mas sim uma gravação que nos apresenta opções de escolha e, consoante as nossas escolhas nos leva ao destino. Em sites *online* também está disponível o *chatbot* que consiste em tentar ajudar o cliente analisando as suas perguntas via textual, estes analisam as palavras e tentam dar uma resposta, se não for possível encaminham para um funcionário. Um bom exemplo da aplicação de *Deep Learning* no atendimento ao cliente são as assistentes virtuais da Google e da Apple, que realizam ações e comandos com base no reconhecimento da voz.

- Saúde: o ramo da saúde evoluiu bastante com a aplicação do *Deep Learning* pois é possível analisar imagens e detetar padrões nestas, ajudando assim os médicos e radiologistas a traçar um diagnóstico mais rapidamente. [12]

Redes Neuronais Convolucionais

A ideia por trás de uma rede neuronal convolucional é filtrar as imagens antes de treinar a rede neuronal profunda. Depois de filtrar as imagens, as características delas podem ser destacadas e podemos localizar as características para identificar alguma coisa.

Um filtro é simplesmente um conjunto de multiplicadores, ou seja, aplicar um filtro a uma imagem é multiplicar o valor da matriz do filtro pela coluna e linha correspondente na imagem, dando origem a novos valores em cada pixel da imagem. Desta forma consegue-se destacar alguns aspetos da imagem em questão.

Isto pode ainda ser combinado com uma camada *Max Pool* num processo chamado de *pooling* que agrupa os pixéis da imagem e os filtra num subconjunto. Por exemplo usando o *max pooling* de 2x2 a imagem será agrupada em conjuntos de 2x2 pixéis e simplesmente escolhe o maior de entre os 4 e desta forma a imagem será reduzida a um quarto do seu tamanho original, mas as características mais relevantes serão mantidas e até realçadas.

Numa camada convolucional vários filtros são inicializados aleatoriamente serão aplicados sobre a imagem. Os resultados passam para a próxima camada e a associação será executada pela rede neuronal e ao longo do tempo os filtros que produzirem as imagens de saída com as melhores associações serão aprendidos e o processo é chamada de extração de características.

O primeiro parâmetro da Conv2D (camada convolucional 2D) corresponde ao número de filtros que vão passar pela imagem e a cada *epoch*, ou seja, a cada ciclo de treino, vai

identificar quais filtros produziram os melhores sinais para ajudar a associar as imagens com os seus rótulos. Estes filtros têm um tamanho denominado de *kernel size* que correspondem ao segundo parâmetro da função.

O filtro é movido pela imagem da esquerda para a direita, de cima para baixo, com uma alteração de coluna de um pixel nos movimentos horizontais e, em seguida, uma alteração de linha de um pixel nos movimentos verticais.

A quantidade de movimento entre as aplicações do filtro sobre a imagem de entrada é chamada de *stride* e é quase sempre simétrica nas dimensões de altura e largura.

O *stride* padrão ou *strides* em duas dimensões é (1,1), neste caso significa que o filtro é movido 1 pixel de cada vez, para o movimento de altura e largura, realizado quando necessário. E esse padrão funciona bem na maioria dos casos.

O *stride* pode ser alterado, o que afeta tanto a forma como o filtro é aplicado à imagem como também o *output* que resultante. Por exemplo para um filtro com *stride* de tamanho (3,3) os 9 pixéis contidos nele serão convertidos para apenas 1 pixel, ou seja, quanto maior este valor, mais pequeno será o *output* final face ao tamanho original.

Por exemplo, o *stride* pode ser alterado para (2,2). Isto tem o efeito de mover o filtro dois pixéis para a direita para cada movimento horizontal do filtro e dois pixéis para baixo para cada movimento vertical do filtro ao criar o *output* com o mapa de características.

Por padrão, um filtro começa à esquerda da imagem com o lado esquerdo do filtro nos pixéis mais à esquerda da imagem. O filtro é então colocado na imagem, uma coluna de cada vez, até que o lado direito do filtro esteja posicionado nos pixéis mais à direita da imagem.

Uma abordagem alternativa para aplicar um filtro a uma imagem é garantir que cada pixel na imagem tenha a oportunidade de estar no centro do filtro.

Por padrão, esse não é o caso, pois os pixéis na borda da entrada são expostos apenas na borda do filtro. Ao iniciar o filtro fora do *frame* da imagem, ele dá aos pixéis na borda da imagem mais uma oportunidade de interagir com o filtro e mais uma oportunidade para as características mais importantes serem detetadas pelo filtro e, por sua vez, uma saída mapa de características que tem a mesma forma que a imagem de entrada.

No *Keras* [24], uma API, *Application Programming Interface*, de *deep learning* escrita em Python que é executada na plataforma de *machine learning* Tensorflow, isso é especificado por meio do argumento *padding* na camada Conv2D, que tem o valor padrão de *valid* (sem preenchimento). Isso significa que o filtro é aplicado apenas a formas válidas para a entrada.

O valor *padding* de *same* calcula e adiciona o preenchimento necessário à imagem de entrada (ou mapa de características) para garantir que a saída tenha a mesma forma que a entrada [15].

Efficient Net

As redes neuronais convolucionais (CNNs) são normalmente desenvolvidas a um custo fixo de recursos e, de seguida, ampliadas para obter melhor precisão quando são disponibilizados mais recursos. Por exemplo, o ResNet [25] pode ser ampliado de ResNet-18 para ResNet-200 aumentando o número de camadas. A prática mais convencional para o dimensionamento de um modelo é aumentar arbitrariamente a profundidade ou largura da CNN, ou usar uma maior resolução das imagens de input para treinar e avaliar. Embora estes métodos melhorem de facto a precisão, requerem um ajuste manual entediante e ainda assim produzem um desempenho abaixo do ideal.

No artigo da *International Conference on Machine Learning* de 2019, *EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks*, foi proposto um novo método de dimensionamento de modelos que usa um coeficiente composto simples, mas altamente eficaz para aumentar as CNNs de uma maneira mais estruturada. Ao contrário das abordagens convencionais que dimensionam arbitrariamente as dimensões da rede, como largura, profundidade e resolução, este método dimensiona uniformemente cada uma destas dimensões da rede com um conjunto fixo de coeficientes de dimensionamento. Impulsionados por este novo método de dimensionamento e pelo progresso recente no *Auto Machine Learning*, foi então desenvolvida uma família de modelos chamada *EfficientNets* que supera a precisão de última geração com eficiência até 10 vezes melhor (menor e mais rápida) [20].

Dimensionamento de modelos composto: uma maneira melhor de ampliar as CNNs

Para entender o efeito do dimensionamento da rede, foi sistematicamente estudado o impacto do dimensionamento das diferentes dimensões do modelo. Embora o dimensionamento de dimensões individuais melhore o desempenho do modelo, observou-se que equilibrar todas as dimensões da rede – largura, profundidade e resolução da imagem – em relação aos recursos disponíveis melhoraria melhor o desempenho geral.

A primeira etapa no método de dimensionamento composto é realizar uma *Grid Search* para encontrar a relação entre as diferentes dimensões de dimensionamento (Figura 4) da rede de base sob uma restrição de recursos fixos. Isto determina o coeficiente de dimensionamento apropriado para cada uma das dimensões mencionadas anteriormente. De seguida aplicam-se esses coeficientes para ampliar a rede base para atingir o tamanho do modelo desejado ou até alcançar o orçamento computacional [21].

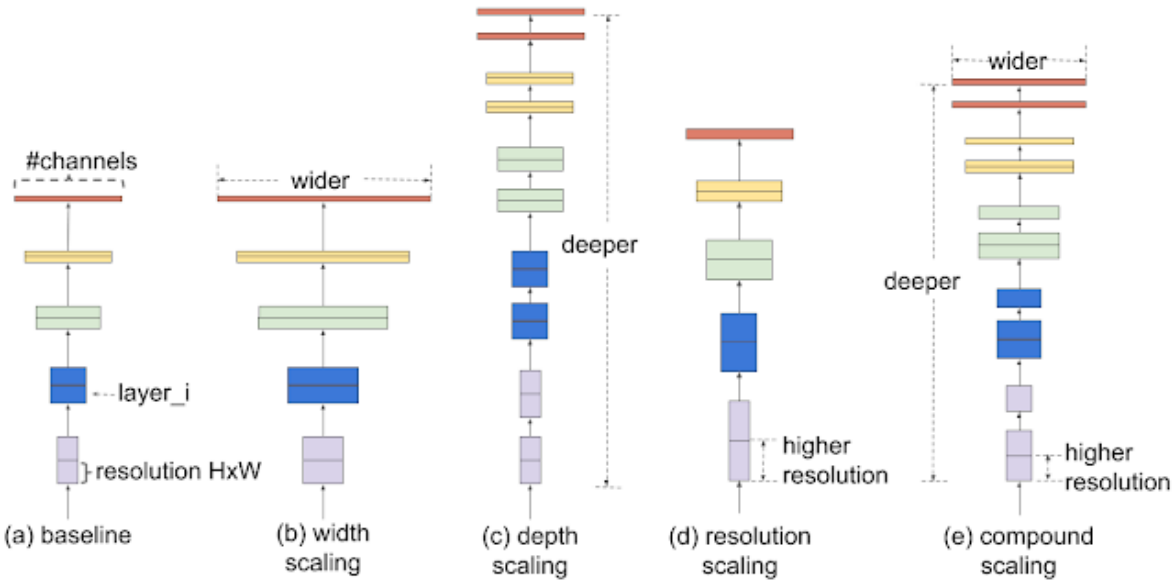

Figura 4 – Comparação de diferentes dimensões de dimensionamento

Arquitetura

O *EfficientNet* [26] é baseado na rede de base desenvolvida pela pesquisa de arquitetura neuronal usando o *framework AutoML MNAS*. A rede é ajustada para obter a máxima precisão, mas também é penalizada se a rede for muito pesada computacionalmente. Também é penalizado pelo tempo de inferência lento quando a rede leva muito tempo para fazer previsões. A arquitetura usa uma convolução de gargalo invertido móvel semelhante ao *MobileNet V2* [27], mas é muito maior devido ao aumento de *FLOPS - floating point operations per second*. Em seguida, é ampliada a rede de base para obter uma família de modelos, chamada *EfficientNets* [21].

Desempenho

O modelo *EfficientNet* foi comparado com outras CNNs existentes (Figura 5) treinadas com a base de dados *ImageNet*. No geral, os modelos *EfficientNet* alcançam maior

precisão e melhor eficiência em relação às CNNs existentes, reduzindo o tamanho do parâmetro e os FLOPS numa ordem de grandeza. Comparado com o ResNet-50 amplamente utilizado, o EfficientNet-B4 usa FLOPS semelhantes, enquanto melhora a precisão top 1 de 76,3% do ResNet-50 para 82,6% (+6,3%) [21].

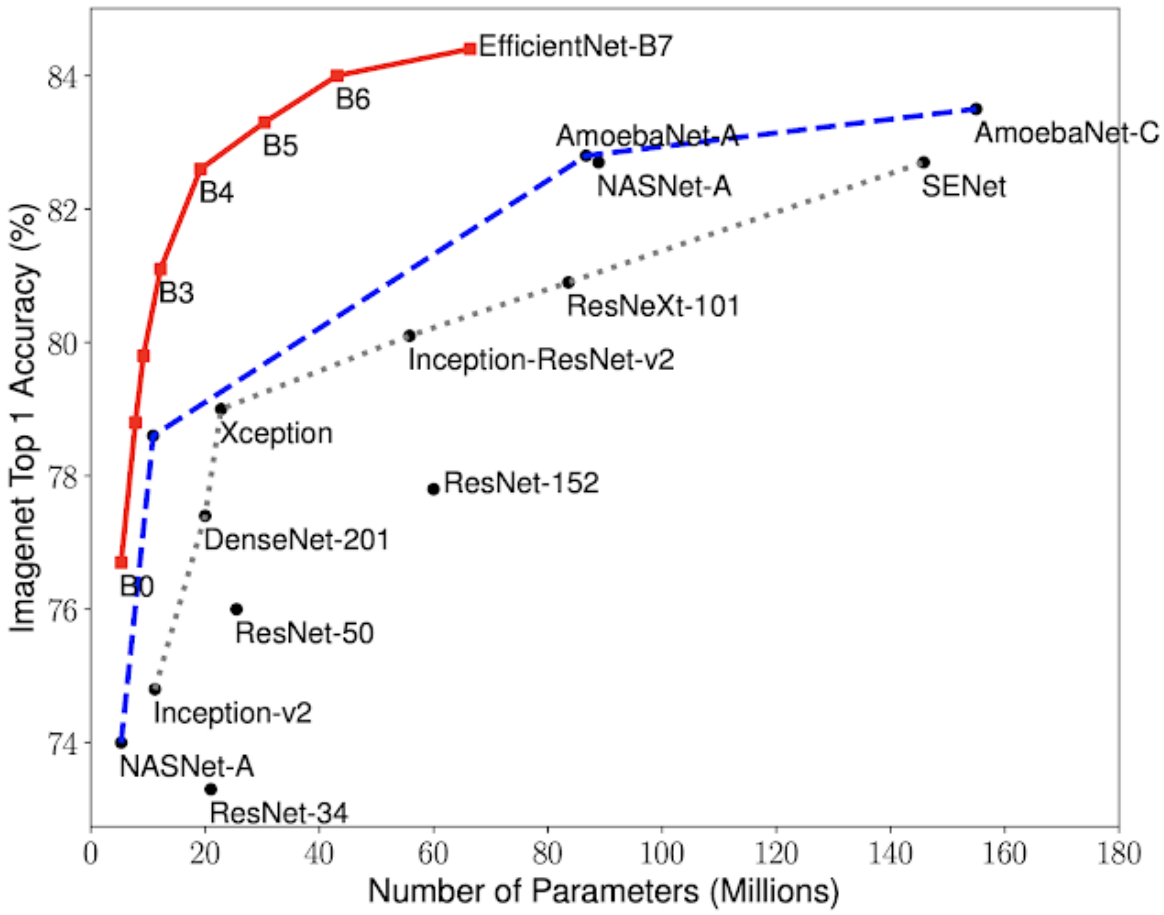

Figura 5 - Comparação do desempenho de vários modelos

Descrição do processo

Dataset utilizado

Para a execução deste projeto foram usadas 6334 imagens de radiografias de pacientes da plataforma *Kaggle* no âmbito da competição "SIIM-FISABIO-RSNA COVID-19 Detection" que foram previamente anotadas com as seguintes categorias por um grupo de radiologistas: negativo para a pneumonia, típico, indeterminado ou atípico para a COVID-19. Das 6334 imagens, 483 correspondem à categoria *Atypical Appearance* (atípico), 1108 à categoria *Indeterminate Appearance* (indeterminado), 3007 à categoria *Typical Appearance* (típico), e 1736 à categoria *Negative for Pneumonia* (negative). Uma vez que as imagens fornecidas pela plataforma para teste não se encontravam anotadas, utilizámos 80% das imagens de treino efetivamente para treinar e os restantes 20% para efeitos de validação do modelo.

Primeira abordagem

Na primeira abordagem da solução para o tema do projeto para o pré-processamento apenas optamos por efetuar uma multiplicação sobre a imagem original de forma a evidenciar certas linhas da mesma obtendo as imagens o seguinte aspeto (Figura 6):

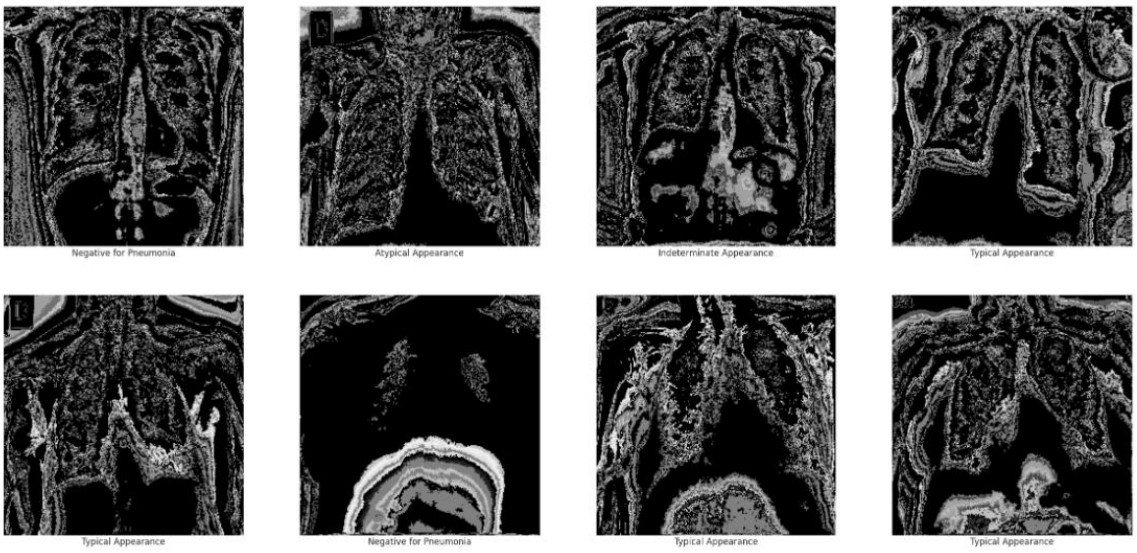

Figura 6 – Resultados da multiplicação sobre a imagem original

Para o modelo em si usámos três blocos de uma camada convolucional, uma camada de *Max Pooling* e uma de *Batch Normalization*.

No primeiro bloco a camada convolucional é constituída por 25 filtros, o tamanho do *kernel* é (5,5), a função de ativação usada é ReLU, o *stride* tem o tamanho de (1,1) e o *padding* tem o valor "same". A camada de *max pooling* tem um valor de (2,2) e *padding* "same".

No segundo bloco a camada convolucional é constituída por 50 filtros, o tamanho do *kernel* é (5,5), a função de ativação usada é ReLU, o *stride* tem o tamanho de (2,2) e o *padding* tem o valor "same". A camada de *max pooling* tem um valor de (2,2) e *padding* "same".

No segundo bloco a camada convolucional é constituída por 70 filtros, o tamanho do *kernel* é (3,3), a função de ativação usada é ReLU, o *stride* tem o tamanho de (2,2) e o *padding* tem o valor "same". A camada de *max pooling* tem um valor de (2,2) e *padding* "valid".

De seguida foi aplicada uma camada *Flatten* e duas *Dense* com o número de unidades com o valor de 100 e a função de ativação ReLU.

Contextualizando, na camada *Flatten* a matriz resultante das camadas anteriores de convoluções e poolings é redimensionada para se tornar um array linear, de uma única dimensão. Esta etapa é um preparo para se entrar na camada principal da rede neuronal totalmente conectada que corresponde à camada *Dense*, e nesta deve-se informar a dimensão da saída e a função de ativação a ser utilizada. No contexto de reconhecimento de imagens é comum se projetar esta camada densa com a função de ativação ReLU e, por fim, outra camada densa com a dimensão igual a quantidade de classes a serem classificadas com a função de ativação *softmax* (para múltiplas classes) [23].

Uma camada de *Dropout* foi considerada a seguir também com o valor de 0.0025.

Por fim a camada de *output* tem um valor de 4 unidades que correspondem às 4 classes em que as imagens do *dataset* do nosso problema podem ser classificadas.

Para compilar o modelo foi usado o *optimizer* "adam", e para a *loss* "sparse_categorical_crossentropy" e como métrica a *accuracy*.

No *fitting* do modelo o *batch_size* tinha o valor de 64, e o modelo foi treinado durante 25 *epochs* usando um *validation split* de 0.2.

Segunda abordagem

Na segunda abordagem que efetuámos para tentar melhorar a previsão do nosso modelo tentámos evidenciar características também, mas desta vez além de efetuar uma multiplicação da imagem por si mesma, aplicámos um filtro que tenta delimitar a área da radiografia que corresponde aos pulmões e fazendo um zoom de forma a cortar partes da periferia da radiografia que poderiam ser menos interessantes para o resultado pretendido. O resultado que obtivemos depois de efetuar esse processo foi o seguinte (Figura 7):

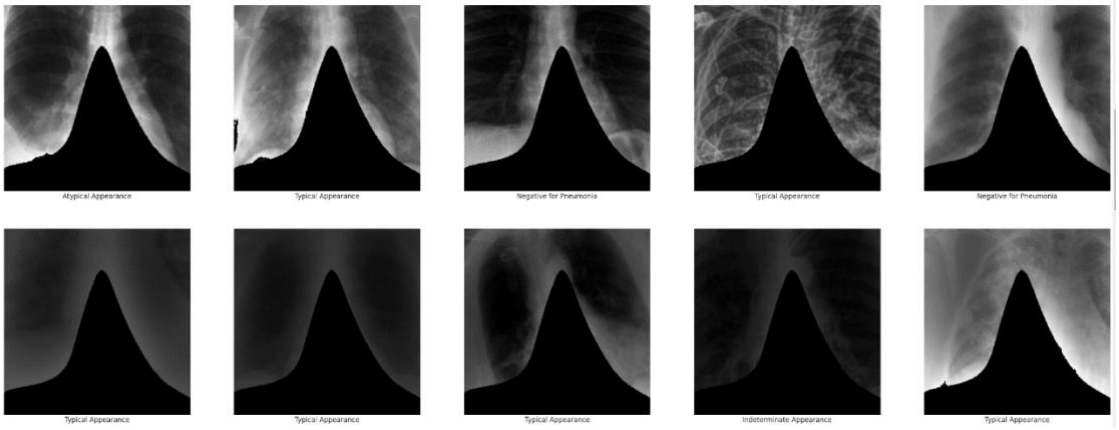

Figura 7 – Resultados da multiplicação, aplicação do filtro e do zoom da imagem

Para chegar ao filtro pretendido efetuou-se a média de todas as imagens e nas regiões em que os valores do pixel fossem superiores a 0.62 colocámos o valor do pixel a 0 de forma que a região ficasse totalmente escura criando a delimitação em redor do formato dos pulmões.

A nível da construção do modelo apenas foram feitas poucas alterações relativamente ao modelo anterior, mantendo-se todos os parâmetros que já tinham sido utilizados exceto na antepenúltima e penúltima camada *Dense* em que usámos um valor de unidades superior, 128. A camada de *Dropout* também sofreu uma ligeira alteração na medida em que aumentámos o seu valor para 0.25. No *fitting* do modelo também reduzimos o *batch_size* para 32.

Terceira abordagem

Numa nova tentativa de melhorar os resultados obtidos optámos por acrescentar um pedaço de código ao que já tínhamos anteriormente que permitisse centrar os pulmões de cada uma das radiografias.

Para chegar a este resultado o nosso algoritmo para esse fim começa por calcular a correlação entre a radiografia em questão e a média de todas as radiografias que nos vai indicar as coordenadas onde a correlação é máxima.

A partir dessa informação subtraímos ao valor de x e y metade do tamanho da imagem original e conseguimos saber a deslocação que terá de ser feita para centrar a radiografia relativamente à média.

De forma a otimizar a execução de todo o processo de centrar as imagens reduzimos o tamanho das imagens a um quarto do tamanho original passando do tamanho de 256x256 para 64x64 e depois de feito o alinhamento dos pulmões ao centro cortamos 5% da imagem com o objetivo de eliminar dados da periferia da radiografia que poderiam ser menos interessantes para o fim pretendido.

Desta forma conseguimos então fortalecer o pré-processamento dos dados e de seguida aplicámos exatamente o mesmo modelo que usámos nas abordagens anteriores.

Quarta abordagem – abordagem final

Nesta última abordagem mantivemos o pré-processamento das imagens que já tínhamos usados anteriormente e apenas substituímos o modelo a usar, optando pelo *EfficientNet* com os pesos pré-treinados do *dataset ImageNet*.

Apesar de ser bastante semelhante às abordagens anteriores, dada a relevância deste código explicamos de seguida com maior detalhe (Figura 8 a 16):

Primeiro atribuímos o nome do diretório onde se encontram todas as imagens a ser utilizadas a uma variável.

Através da biblioteca *pandas* abrimos o ficheiro csv "train_study_level.csv" que contém a informação relevante relativamente à *label* que corresponde a cada imagem e abrimos também o ficheiro csv "train_meta_256x256.csv" que mapeia o id de cada imagem com o *study_id* do ficheiro anteriormente aberto.

Relativamente ao primeiro csv criamos uma nova coluna com o valor da *label*, se for *Negative for Pneumonia* fica com o valor 0, *Typical Appearance* fica com o valor 1, *Indeterminate Appearance* com o valor 2 e *Atypical Appearance* com o valor 3 e posteriormente fazemos o *merge* dos dois *dataframes* num só.

```
DIRECTORY = r'/content/drive/MyDrive'
CATEGORY = ["train_256x256"]

output = []

for category in CATEGORY:
    path = os.path.join(DIRECTORY, category)
    images = []
    labels = []
    imgs = []
    ims = []
    print("Loading ".format(category))

    study_df = pd.read_csv (r'/content/drive/MyDrive/train_study_level.csv')
    df = pd.read_csv (r'/content/drive/MyDrive/train_meta_256x256.csv')

    def value (row):
        if row['Negative for Pneumonia'] == 1 :
            return '0'
        if row['Typical Appearance'] == 1 :
            return '1'
        if row['Indeterminate Appearance'] == 1:
            return '2'
        if row['Atypical Appearance']  == 1:
            return '3'
        return '4'

    study_df['value'] = study_df.apply (lambda row: value(row), axis=1)


    study_df = study_df.rename({'id':'study_id'}, axis=1)
    df['study_id'] = df['study_id']+ '_study'
    df = pd.merge(df, study_df, on = 'study_id', how = 'left')
```

Figura 8 – Código da quarta abordagem

Com a função *listdir* da biblioteca *os* colocamos todos os ficheiros da pasta *path* na variável l e com a função *shuffle* da biblioteca *random* ordenamos esses ficheiros de forma aleatória.

Através de um ciclo *for* percorremos todos os elementos da lista l que correspondem às imagens do *dataset* e retiramos o valor da sua *label* que posteriormente fazemos o *append* ao *array* “labels”

Se o valor da variável *categorias equilibradas* estiver como valor “True” apenas recolhemos o mesmo número de imagens de cada classe e fazemos a sua contagem, caso contrário e o seu valor seja “False” apenas fazemos a contagem de quantas imagens existem por categoria através das variáveis criadas para o efeito que correspondem às iniciais de cada categoria.

```
#todos os ficheiros dentro da pasta
l = os.listdir(path)
#ordená-los de forma aleatória
random.shuffle(l)

h=0
for file in l:
    if h < Maxdim:
        h+=1
        #contagem de número de imagens para cada classe
        global IA
        global TA
        global AA
        global NP

        value = df.loc[df['id'] == file.replace('.png',''),'value'].values[0]

        if categorias_equilibradas:
            #equilibrar o número de imagens para cada classe
            if value == '0' and NP < nImagensClasse:
                NP+=1
            elif value == '1' and TA < nImagensClasse:
                TA+=1
            elif value == '2' and IA < nImagensClasse:
                IA+=1
            elif value == '3' and AA < nImagensClasse:
                AA+=1
            else:
                continue
        else:
            #equilibrar o número de imagens para cada classe
            if value == '0':
                NP+=1
            elif value == '1':
                TA+=1
            elif value == '2':
                IA+=1
            elif value == '3':
                AA+=1
            else:
                continue

        labels.append(value)
```

Figura 9 – Código da quarta abordagem (continuação)

De seguida através da biblioteca *cv2* e da função *imread* passando o *path* da imagem como parâmetro abrimos a imagem e fazemos o redimensionamento da imagem através da função *resize* da mesma biblioteca para o tamanho 224*224.

Depois disso atribuímos à variável *imageMedia* o valor do cálculo presente no código e somamos esse valor à soma de todas as imagens e à contagem das mesmas de forma a efetuar a média de todas mais à frente.

Através da função *array* da biblioteca *numpy* convertemos a lista de imagens e de *labels* num *array*.

```
            # Get the path name of the image
            img_path = os.path.join(path, file)


            # Open and resize the img
            image = cv2.imread(img_path)
            image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
            image = cv2.resize(image, IMAGE_SIZE)
            imageMedia = image*(1-0.000003*image*image+0.000000031*image*image*image+0.00000000031*image*image*image*image)

            if cont < Maxdim:
                soma += imageMedia
                cont += 1

            # Append the image and its corresponding Label to the output
            images.append(image)

images = images[0: Maxdim]
labels = labels[0: Maxdim]
images = np.array(images, dtype = 'float32')
labels = np.array(labels, dtype = 'int32')

print("np.shape(images)=", np.shape(images))
media = soma/cont
media /=np.amax(media)
print("imagem media =", media)
```

Figura 10 – Código da quarta abordagem (continuação)

O próximo passo foi tentar centrar os pulmões das imagens em comparação com a média das imagens. Para conseguir este efeito percorremos todas as imagens do *dataset* num ciclo *for*.

Começámos por reduzir o tamanho da imagem média e de a colocar num vetor unidimensional com o tamanho das suas dimensões multiplicado por três (que corresponde aos canais RGB) e procedemos da mesma forma para cada uma das imagens do *dataset* previamente integradas no *array* “images”.

Como verificámos que os valores nas 3 camadas RGB de cada imagem tinham o mesmo valor voltamos a fazer o mesmo processo, mas descartando então duas delas colocando ambas as imagens num *array* das suas dimensões.

```
TempMedia = np.reshape(media, [dim*dim*3])

for n in range(0, Maxdim):
    media_ = cv2.resize(media, IMAGE_SIZE2)
    TempMedia1D = np.reshape(media_, [dim2*dim2*3])
    images_n = cv2.resize(images[n], IMAGE_SIZE2)
    TempImg1D = np.reshape(images_n, [dim2*dim2*3])

    TempImg = []
    TempMedia2 = []

    for m in range(0, dim2*dim2):
        TempImg.append(TempImg1D[3*m])
        TempMedia2.append(TempMedia1D[3*m])


    media2D = np.reshape(TempMedia2, [dim2,dim2])
    Img2D = np.reshape(TempImg, [dim2,dim2])
```

Figura 11 – Código da quarta abordagem (continuação)

De seguida com recurso à biblioteca *SciPy* e à função *signal.correlate2d* calculamos a correlação entre a média e a imagem atual com a parâmetro "mode" com o valor "same" para que o *output* seja do mesmo tamanho que o primeiro *array*, e com o parâmetro "boundary" com o valor "wrap" e guardamos o *output* no *array* corr2D.

Depois disso com recurso à biblioteca *numpy* e à função *where* obtemos os índices do *array corr2D* onde o seu valor é o máximo e de seguida com a função *roll* fazemos o *shift* da imagem no número de elementos correspondente à diferença entre o índice onde a correlação é máxima e o centro da imagem, tanto para o eixo vertical como para o horizontal e posteriormente cortamos a imagem em 5%, em cada uma das 4 bordas para fazer um pequeno zoom de forma a retirar partes menos relevantes. A seguir a correlação é efetuada novamente só como verificação de que efetivamente foi bem feita.

```
corr2D = signal.correlate2d (media2D, Img2D, boundary='wrap', mode='same')
#print("signal.correlate2d (media2D, Img2D)=", corr2D)

#print("Valor da correlação máxima corr2D.argmax()=", corr2D.argmax())
#print("Coordenadas do valor máximo da Coorelação2D:", np.unravel_index(np.argmax(corr2D), corr2D.shape))
#print("corr2D[np.unravel_index(np.argmax(corr2D), corr2D.shape)]=", corr2D[np.unravel_index(np.argmax(corr2D), corr2D.shape)])

indices = np.where(corr2D == corr2D.max())
#print("indices onde a correlação é máxima",indices)
#print("indices[0]=",indices[0])
#print("indices[1]=",indices[1])
#print("corr2D[indices]=",corr2D[indices])
#print("corr2D[indices[0], indices[1]]=",corr2D[indices[0], indices[1]])

images[n] = np.roll(images[n], int(dim/dim2)*indices[0]-(int(dim/2)), axis=0) # shift 1 place in horizontal axis
images[n] = np.roll(images[n], int(dim/dim2)*indices[1]-(int(dim/2)), axis=1) # shift 1 place in vertical axis
#print("translação indices[0]=",int(dim/dim2)*indices[0]-(int(dim/2)))
#print("translação indices[1]=",int(dim/dim2)*indices[1]-(int(dim/2)))
image = images[n]
#corte 5% na imagem
image = image[int(0.05*dim):int(0.95*dim), int(0.05*dim):int(0.95*dim)]
images[n] = cv2.resize(image, IMAGE_SIZE)

image_n_tranformada = cv2.resize(images[n], IMAGE_SIZE2)
TempImg1DTransformada = np.reshape(image_n_tranformada, [dim2*dim2*3])
ImgTransformada = []

for m in range(0, dim2*dim2):
    ImgTransformada.append(TempImg1DTransformada[3*m])

Img2DTransformada = np.reshape(TempImg, [dim2,dim2])

#print("imagem translatada para centrar correlação2D em (127,127)=")
#plt.figure()
#plt.imshow(Img2DTransformada)
#plt.show()

corr2D = signal.correlate2d (media2D, Img2DTransformada, boundary='wrap', mode='same')
#print("Coordenadas do valor máximo da Coorelação2D após translação:", np.unravel_index(np.argmax(corr2D), corr2D.shape))
```

Figura 12 – Código da quarta abordagem (continuação)

Posteriormente as imagens e respetivas *labels* são mais uma vez baralhadas dentro dos seus *arrays* e depois é feito o particionamento entre as imagens de treino e de teste.

```
images, labels = shuffle(images, labels, random_state=25)


(train_images, train_labels) = (images[0:limitForTest], labels[0:limitForTest])
(test_images, test_labels) = (images[limitForTest:Maxdim], labels[limitForTest:Maxdim])

print(len(train_images))
print(len(train_labels))
print(len(test_images))
print(len(test_labels))
```

Figura 13 – Código da quarta abordagem (continuação)

Para treinar o modelo usámos o *EfficientNetB0*, uma aplicação da *API Keras* do *Tensorflow* pré-treinada com pesos na base de dados *ImageNet*.

Para compilar o modelo usámos o *optimizer* "Adam", para *loss* "Sparse Categorical Crossentropy" e para *mettrics* "Accuracy"

De seguida é feito o *fit* do modelo com as imagens de treino, o *batch size* a 64, *epochs* com o valor 10, e *validation split* de 0.2; isto devolve um objeto *History* que é guardado na variável *history*.

O modelo depois é guardado localmente para ser usado posteriormente.

```
model.summary()

model.compile(optimizer = 'adam', loss = 'sparse_categorical_crossentropy', metrics=['accuracy'])


history = model.fit(train_images, train_labels, batch_size=64, epochs=15, validation_split = 0.2)
```

Figura 14 – Código da quarta abordagem (continuação)

Na função *plot_accuracy_loss* passamos a variável *history* para elaborar um gráfico que mostra a evolução da *accuracy* e da *loss* ao longo das *epochs*.

```
def plot_accuracy_loss (history):
    """
    Plot the accuracy and the loss during the training of the nn.
    """
    fig = plt.figure(figsize=(10,5))
    # Plot accuracy
    plt.subplot(221)
    plt.plot(history.history['accuracy'], 'bo--', label = "acc")
    plt.plot(history.history['val_accuracy'], 'ro--', label = "val_acc")
    plt.title("train_acc vs val_acc")
    plt.ylabel("accuracy")
    plt.xlabel("epochs")
    plt.legend()

    # Plot loss function
    plt. subplot(222)
    plt.plot(history.history['loss'],'bo--', label = "loss")
    plt.plot(history.history['val_loss'], 'ro--', label = "val_loss")
    plt.title("train_loss vs val_loss")
    plt.ylabel("loss")
    plt.xlabel("epochs")

    plt.legend()
    plt.show()

plot_accuracy_loss(history)
```

Figura 15 – Código da quarta abordagem (continuação)

A seguir é feita a *prediction* com as *test_images* e o resultado é guardado na variável *predictions e* fazemos uma matriz de confusão para melhor interpretar os dados.

```
test_loss = model.evaluate(test_images, test_labels)
test_labels=test_labels

predictions = model.predict(test_images)
pred_labels = np.argmax(predictions, axis = 1)
print(test_labels)
print(pred_labels)
print(classification_report(test_labels, pred_labels))



#Get the confusion matrix
cf_matrix = confusion_matrix(test_labels, pred_labels)

print(cf_matrix)
group_counts = ["{0:0.0f}".format(value) for value in
cf_matrix.flatten()]
group_percentages = ["{0:.2%}".format(value) for value in
cf_matrix.flatten()/np.sum(cf_matrix)]
labels = [f"{v1}\n{v2}\n" for v1, v2 in
zip(group_counts,group_percentages)]
labels = np.asarray(labels).reshape(4,4)
ax = sn.heatmap(cf_matrix, annot=labels, fmt='', cmap='Blues')
ax.set_title('Seaborn Confusion Matrix with labels\n\n');
ax.set_xlabel('\nPredicted Category')
ax.set_ylabel('Actual Category ');
## Ticket labels - List must be in alphabetical order
ax.xaxis.set_ticklabels(['NP', 'TA', 'IA', 'AA'])
ax.yaxis.set_ticklabels(['NP', 'TA', 'IA', 'AA'])
## Display the visualization of the Confusion Matrix.
plt.show()
```

Figura 16 – Código da quarta abordagem (continuação)

Aplicação desenvolvida para aplicar o modelo

Introdução

Para dar sentido ao modelo previamente criado e treinado com o *dataset* fornecido, elaborámos uma aplicação simples *desktop* em Python com uma interface gráfica que permite auxiliar ou complementar o trabalho dos profissionais de saúde no que toca a tirar ilações sobre as radiografias.

Sucintamente, esta aplicação dá a possibilidade de o utilizador selecionar uma imagem única (Figura 17) ou uma pasta que contenha várias imagens de radiografias (Figura 18) e obter a previsão para cada uma delas de acordo com o modelo elaborado, dentro das quatro classes que ele está treinado para prever.

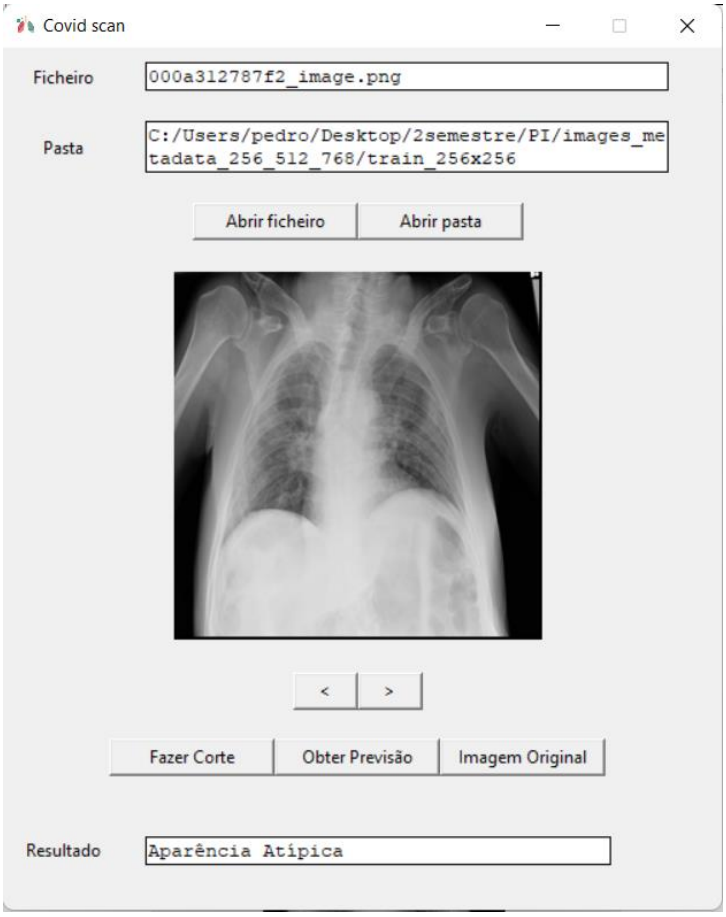

Figura 17 – Aplicação quando se seleciona uma pasta

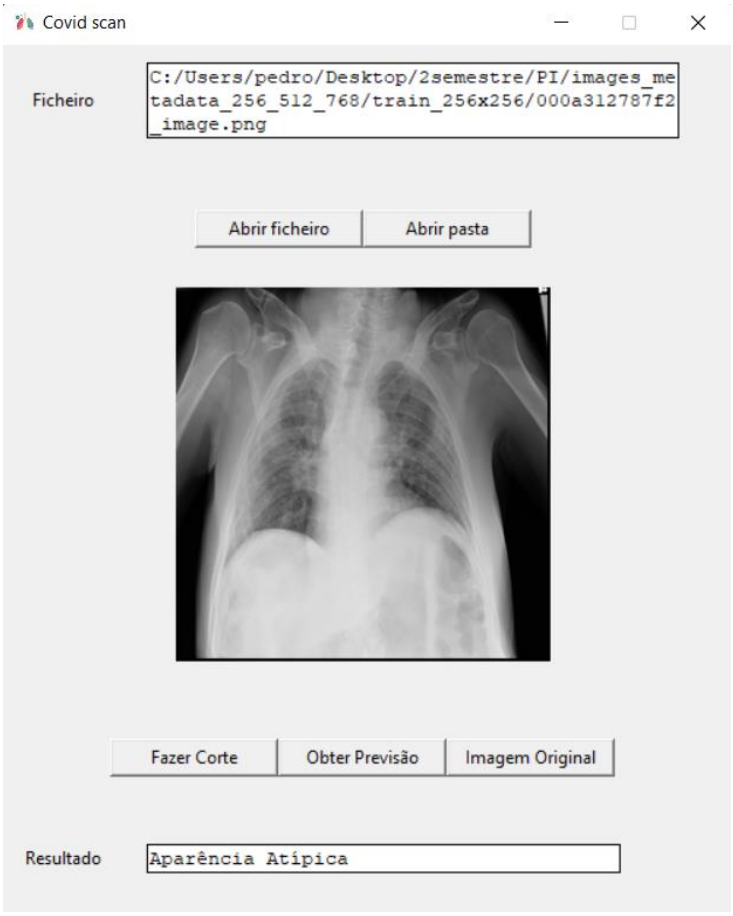

Figura 18 - Aplicação quando se seleciona uma imagem única

Elaboração

Para elaborar esta aplicação recorremos ao *Tkinter* a biblioteca para Interface Gráfica de Utilizador padrão para Python.

Para começar definimos os valores da altura e largura da aplicação e fizemos os procedimentos para centrá-la no ecrã do utilizador e de seguida criámos os textos que aparecem nas caixas inicialmente quando o utilizador ainda não selecionou nenhum ficheiro/pasta.

Para selecionar uma pasta implementámos a função *select_folder* (Figura 19 e 20) que através do *filedialog* do *Tkinter* e da função *askdirectory* permite ao utilizador selecionar uma pasta do seu computador para ser aberta. Depois de selecionada, caso seja inválida os botões que permitem a usabilidade da aplicação permanecem desativados, caso contrário são ativados e ainda são adicionados os botões que permitem navegar pelas imagens da pasta. Posteriormente através da função *setFolder* o nome da pasta é apresentado ao utilizador na caixa de texto respetiva.

```
def select_folder():
    global directory
    directory = fd.askdirectory(
        title='Abrir pasta',
        initialdir='/desktop'
    )

    if(directory == ""):
        obterPrevisao_button['state'] = "disabled"
        fazerCorte_button['state'] = "disabled"
        reverter_button['state']="disabled"
        avancar_button['state']="disabled"
    else:
        folderBox.grid(padx=25, sticky=W, row=1, column=1)
        folder.grid(row=1, column=0,  padx=(10,0))
        obterPrevisao_button['state'] = "normal"
        fazerCorte_button['state'] = "normal"
        reverter_button['state'] = "normal"
        avancar_button['state']="normal"
        avancar_button.grid(row=0, column=1)
        recuar_button.grid(row=0, column=0)
        setFolder(directory)
```

Figura 19 - Função select_folder

É criada também a variável *images* que irá guardar todos *filenames* das imagens que se encontram na pasta selecionada pelo utilizador e depois num ciclo *for* para cada *filename* que exista dentro da pasta é testado se corresponde a uma imagem ou não e, em caso afirmativo, este é adicionado à variável *images*.

A primeira imagem é então aberta e enviada para a função *mostrarImagem* com o valor "True" que indica que é a primeira imagem a ser aberta na aplicação e o *filename* da imagem é colocado na respetiva caixa de texto através da função *setFilename*.

```
global images
#guardar os filenmaes
images = []
#Percorrer os filenames do diretório
for filename in os.listdir(directory):
    #Tentar abrir o filename
    imagem = cv2.imread(os.path.join(directory,filename))
    #Se conseguir abrir é uma imagem e por isso guardamos na variável
    if imagem is not None:
        images.append(os.path.join(directory,filename))

#Mostrar a imagem
global imagemAtual, imgOriginal
imagemAtual = 0
img = Image.open(images[imagemAtual]).convert('RGB')
imgOriginal = img
mostrarImagem(img, True)
setFilename(images[imagemAtual].replace(directory+'\\', ''))
```

Figura 20 - Função select_folder (continuação)

A função *mostrarImagem* (Figura 21) é a que permite à aplicação, tal como o nome indica, mostrar a imagem no *canvas* destinado para tal. Caso a imagem seja proveniente de um corte feito anteriormente é feito o redimensionamento para manter o *ratio* feito pelo corte e de seguida passa-se a imagem para a função *make_square* que é responsável por torná-la quadrada e preencher as laterais para ficar com o tamanho correto.

Caso seja a primeira imagem a ser mostrada na aplicação é usado o método *create_image* do *canvas* e caso contrário é usado o método *itemconfig* que nos permite mudar a imagem atualmente em exibição.

```
def mostrarImagem(img, first):
    global imgAAvaliar, imgCanvas, image_container
    #manter o ratio da imagem
    new_height = 224
    new_width  = int(new_height * img.size[0] / img.size[1])
    imgCanvas = img.resize((new_width, new_height),  Image.Resampling.LANCZOS)
    #tornar quadrada e preencher as bordas
    imgAAvaliar = make_square(imgCanvas)
    imgCanvas = ImageTk.PhotoImage(imgAAvaliar)
    #Se for a primeira vez que a aplicação mostra uma imagem canvas.create
    if first == True:
        image_container = canvas.create_image(0,0, image=imgCanvas, anchor='nw')
    #senão canvas.itemconfig
    else:
        canvas.itemconfig(image_container,image=imgCanvas)
```

Figura 21 - Função "mostrarImagem"

Outra função com relevância na elaboração desta aplicação foi a *obterPrevisao* (Figura 22): através da biblioteca *keras* é carregado o modelo previamente guardado, a imagem a avaliar é transformada num *array* e é feito o seu *reshape* para corresponder ao parâmetro de entrada do modelo e a partir daí com o *model.predict* obtém-se a previsão para a imagem em questão. De seguida é feita a correspondência entre o número devolvido e a classe que lhe dá significado e atribui-se esse valor à caixa de texto que contém o resultado.

```
def obterPrevisao():
    model = keras.models.load_model(r'modeloFinal0.6430.h5')
    im_np = np.asarray(imgAAvaliar)
    numpydata = im_np.reshape(1, 224, 224, 3)
    predictions = model.predict(numpydata)
    pred_labels = np.argmax(predictions, axis = 1)

    print("Result")
    print(pred_labels)

    #Resultado
    classe = StringVar()
    #{'Negative for Pneumonia': 0, 'Typical Appearance': 1, 'Indeterminate Appearance': 2, 'Atypical Appearance': 3}
    if(pred_labels[0]==0):
        classe.set("Negativo para a Pneumonia")
        #classe.set("Negative for Pneumonia")
    elif(pred_labels[0]==1):
        #classe.set("Typical Appearance")
        classe.set("Aparência Típica")
    elif(pred_labels[0]==2):
        #classe.set("Indeterminate Appearance")
        classe.set("Aparência Indeterminada")
    else:
        #classe.set("Atypical Appearance")
        classe.set("Aparência Atípica")

    msgResult['state']="normal",
    msgResult.delete(1.0, END)
    msgResult.insert(END, classe.get())
    msgResult['state']="disabled"
```

Figura 22 - Função "obterPrevisao"

Instalação e uso

A aplicação é de fácil utilização uma vez que não requer qualquer tipo de instalação, bastando executar o ficheiro *Python* que se encontra na pasta.

Assim que a aplicação é aberta todas as suas funcionalidades aparecem num só ecrã tornando todo o processo bastante intuitivo.

O primeiro passo é abrir um ficheiro (Figura 23) ou então abrir uma pasta (Figura 24) clicando no respetivo botão. Depois de abrir um ficheiro a imagem é automaticamente apresentada, podendo o utilizador optar por fazer um corte (Figura 25) ou obter a previsão (Figura 26) de classificação para a imagem selecionada.

Figura 23 – Botão "Abrir Ficheiro"

Figura 24 – Botão "Abrir Pasta"

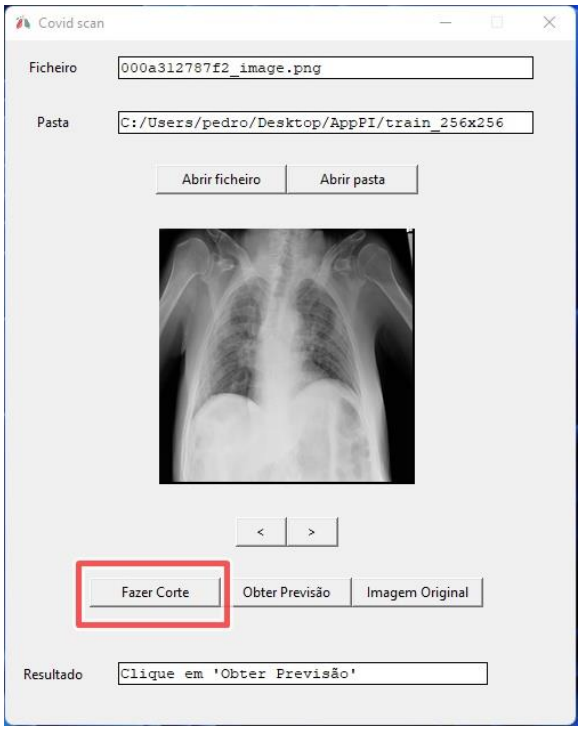

Figura 25 – Botão "Fazer Corte"

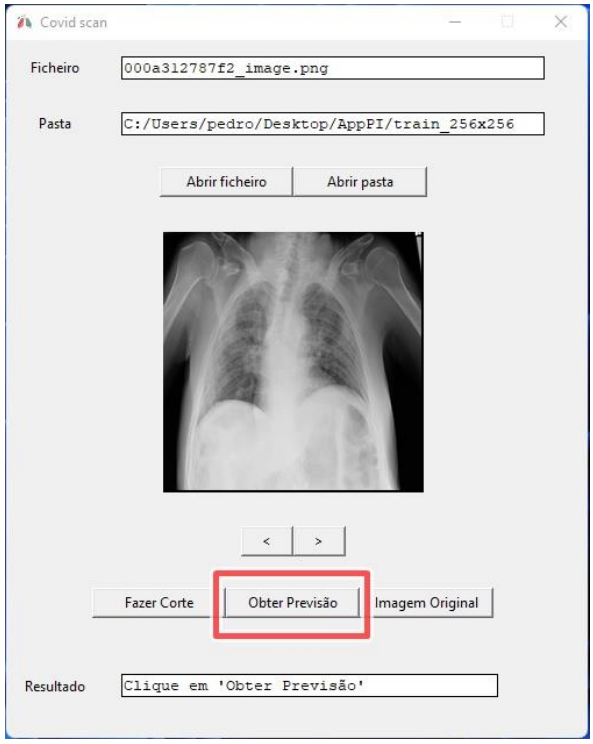

Figura 26 – Botão "Obter Previsão"

Quando o utilizador opta por abrir uma pasta a primeira imagem da pasta é automaticamente apresentada e depois pode usar as setas de navegação para percorrer as restantes imagens da pasta (Figura 27).

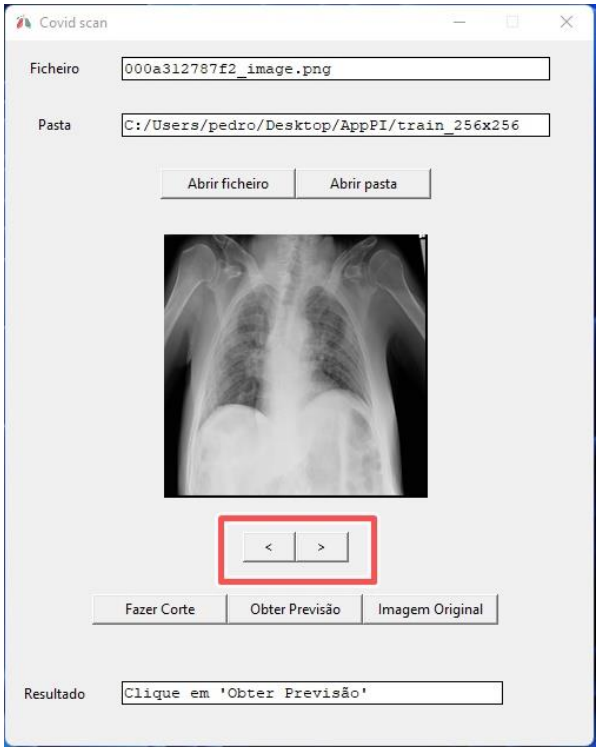

Figura 27 – Botões de navegação

Quando o utilizador seleciona o botão "Fazer Corte" (Figura 25) fica ativa a possibilidade de o fazer diretamente por cima da imagem que lhe é apresentada inicialmente. O corte é feito num formato de caixa bastando o utilizador pressionar o botão esquerdo do rato no sítio mais à esquerda onde pretende começar a caixa e depois, mantendo o botão esquerdo do rato pressionado deslizar para a direita e para baixo, largando quando atinge o tamanho desejado, de maneira a formar a caixa delimitadora que irá resultar na nova imagem cortada.

Depois disso pode carregar no botão "Obter Previsão" e obter uma nova previsão para a imagem atualmente mostrada ou então pode optar por clicar no botão "Imagem Original" (Figura 28) para reverter o corte feito e eventualmente fazer um novo.

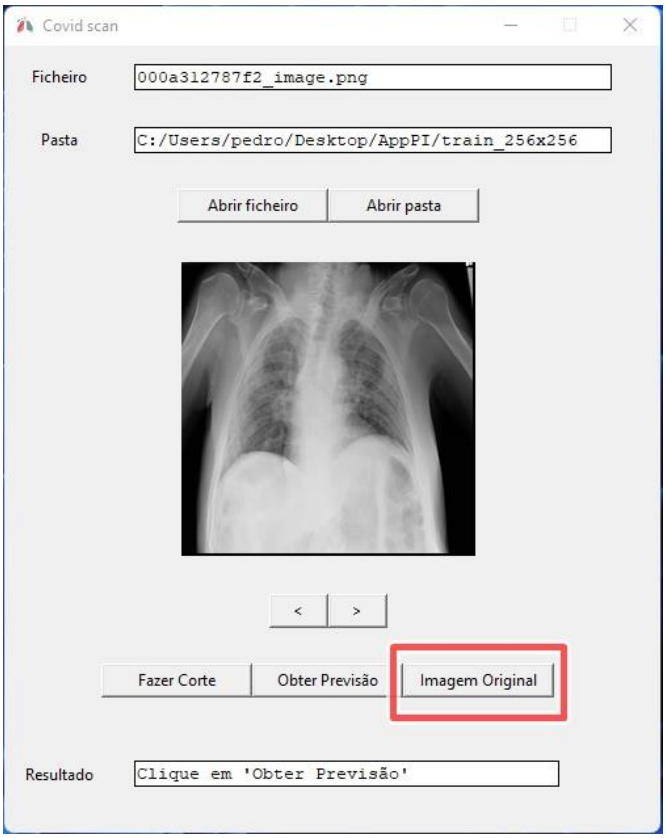

Figura 28 – Botão "Imagem Original"

Posteriormente, recorrendo ao modelo treinado vai ser efetuada a previsão e o resultado será mostrado na caixa de texto abaixo (Figura 29).

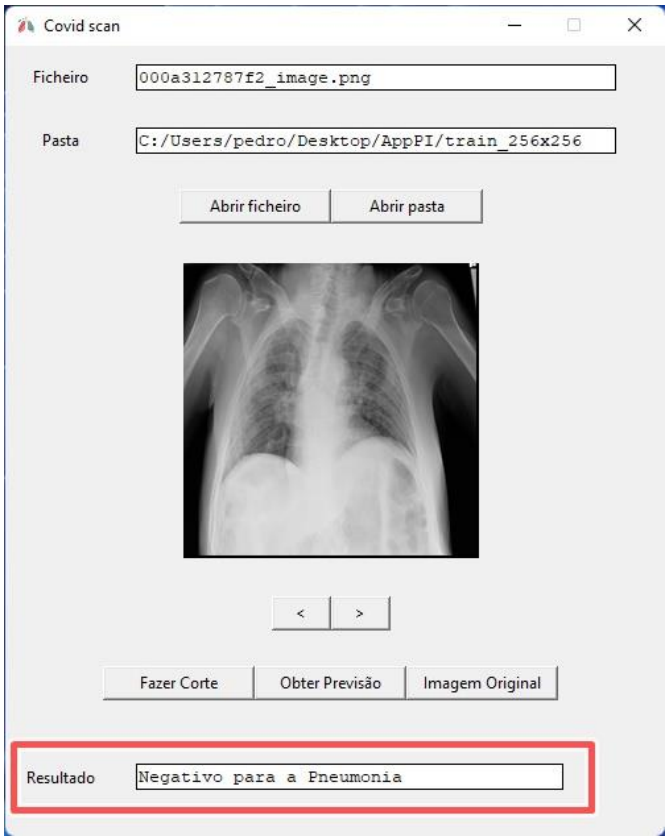

Figura 29 – Caixa de texto com o resultado

Apresentação de resultados

Primeira abordagem

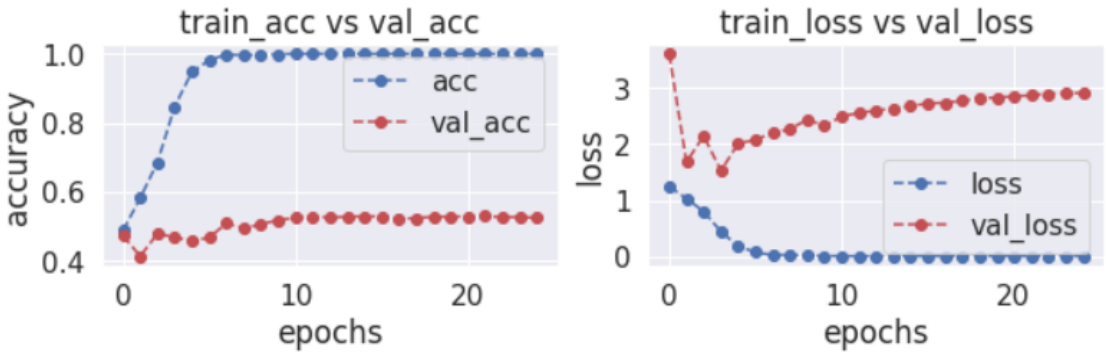

Figura 30 - *Accuracy* e *Loss* ao longo das *epochs* da primeira abordagem

Durante o treino do modelo os resultados crescem gradualmente (Figura 30) até à *epoch* 8 onde tanto a *train accuracy* como a *validation accuracy* acabam por estagnar, ficando a primeira com o valor de 1 e a segunda na casa dos 0.52.

Relativamente à parte reservada nas *test_imagens* o modelo conseguiu prever corretamente apenas 50,62% das imagens, e por isso consideramos que ainda havia espaço por onde melhorar e não estando satisfeitos com estes resultados tentámos uma outra abordagem.

Segunda abordagem

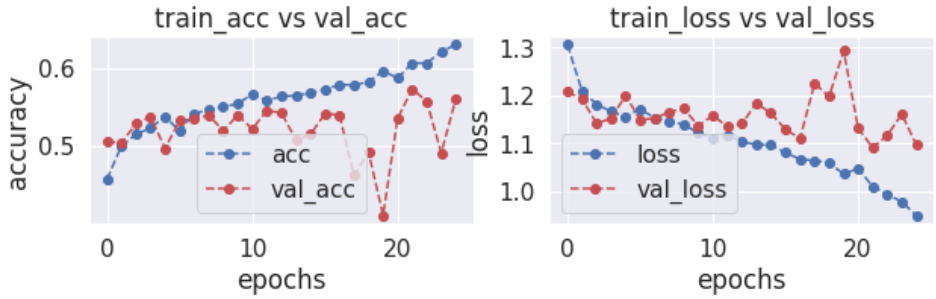

Figura 31 – *Accuracy* e *Loss* ao longo das *epochs* da segunda abordagem

Verificamos nesta segunda abordagem que a *train accuracy* e a *validation accuracy* até certo ponto vão estando próximas (Figura 31), mas chega a uma altura em que acabam por se distanciar e nunca ir além da casa dos 0.5730, atingindo o seu máximo na *epoch* 22.

Relativamente à previsão da parte das imagens que se encontrava nas *test_images* o resultado foi de uma previsão correta de 56.99% do total das imagens, resultando na seguinte matriz de confusão (Figura 32):

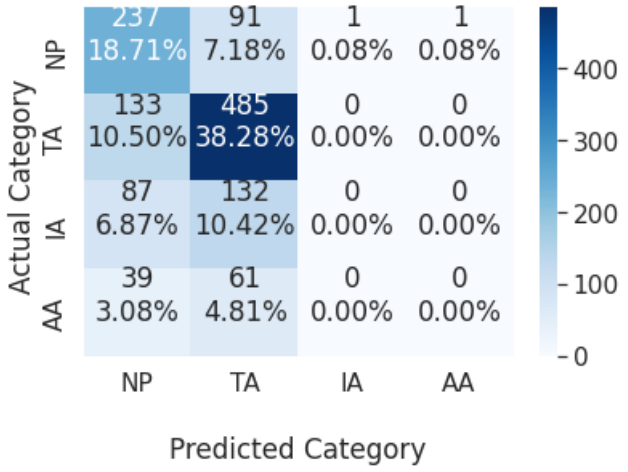

Figura 32 – Matriz de confusão da segunda abordagem

Com estes resultados achámos que ainda havia espaço para melhorar e por isso decidimos avançar com uma outra abordagem.

Terceira abordagem

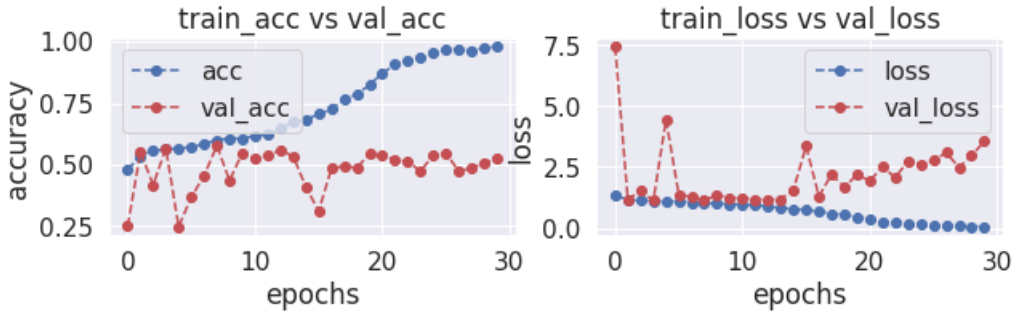

Figura 33 – *Accuracy* e *Loss* ao longo das *epochs* da primeira execução da terceira abordagem

Tal como na abordagem anterior a *train accuracy* e a *validation accuracy* vão crescendo à mesma velocidade até certo ponto (Figura 33), mas a *validation accuracy* atinge o seu pico na *epoch* 8 com o valor de 0.5769, o melhor resultado observado até agora e a partir daí mantém-se mais ou menos constante tendo várias vezes valores a rondar os 0.50.

Quanto às *test_images* o valor também não se distanciou muito obtendo uma previsão correta para 51,62% das imagens (Figura 34).

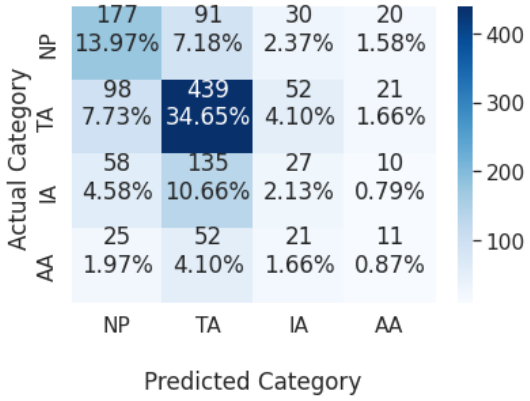

Figura 34 – Matriz de confusão da primeira execução da terceira abordagem

Numa outra tentativa de testes usando esta mesma abordagem conseguimos observar que durante o treino foi atingido o valor 0.6075 na *epoch* 9 (Figura 35), sendo o valor mais alto até então, mantendo-se a performance ao longo das *epochs* seguintes semelhantes ao que já se tinha observado anteriormente (Figura 36).

```
Epoch 9/30
127/127 [==============================] - 150s 1s/step - loss: 1.0453 - accuracy: 0.5801 - val_loss: 1.0954 - val_accuracy: 0.6075
```

Figura 35 – Resultado da *epoch* 9

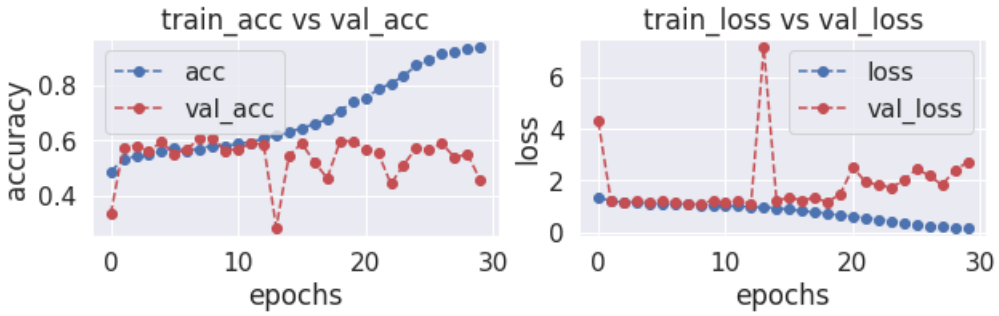

Figura 36 – *Accuracy* e *Loss* ao longo das *epochs* da segunda execução da terceira abordagem

No que diz respeito às *test images* o desempenho não foi o melhor, acertando apenas nas *labels* em 41.28% das imagens e que deu origem à seguinte matriz de confusão (Figura 37):

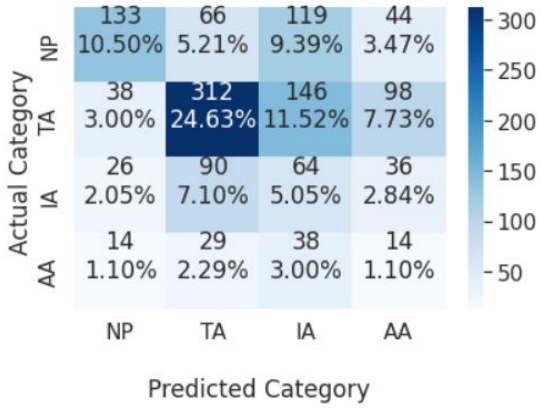

Figura 37 – Matriz de confusão da segunda execução da terceira abordagem

Chegado a estes valores implementámos uma função de *callback* que para de treinar o modelo quando chega ao valor de 0.58 que resultou numa previsão final das *test_images* com o melhor resultado até então, acertando em 59,98% das imagens (Figura 38):

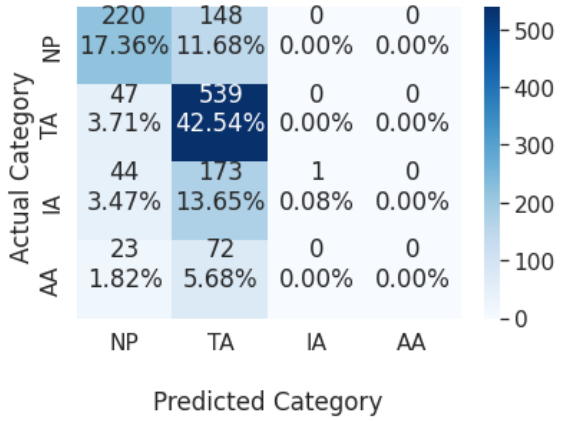

Figura 38 – Matriz de confusão da terceira execução da terceira abordagem

Quarta abordagem – abordagem final

Batch_size 16 e epochs 20

Para vermos até que valores o nosso modelo conseguiria chegar executámos o mesmo cenário, mas sem ativar a função de *callback* e obtivemos o valor de 0.6450 na segunda *epoch* e mais tarde o valor de 0.6321 na quinta *epoch* (Figura 39).

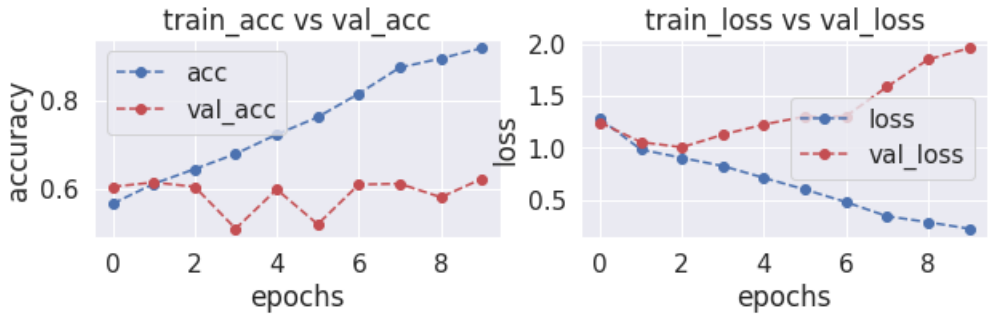

Figura 39 – *Accuracy* e *Loss* ao longo das *epochs* da primeira execução da quarta abordagem

Batch_size 64 e epochs 10

Numa primeira testagem deste cenário obtivemos uma *validation accuracy* de 0.6223 na *epoch* 10 (Figura 40) e nas *test images*, ou seja nas imagens utilizadas para teste, uma taxa de acerto de 61.56% (Figura 41).

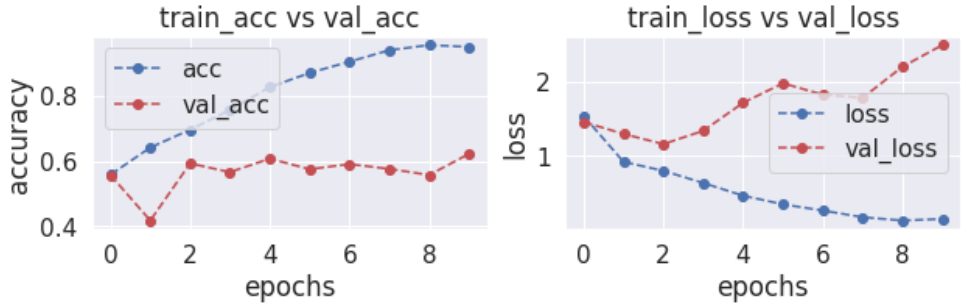

Figura 40 – *Accuracy* e *Loss* ao longo das *epochs* da segunda execução da quarta abordagem

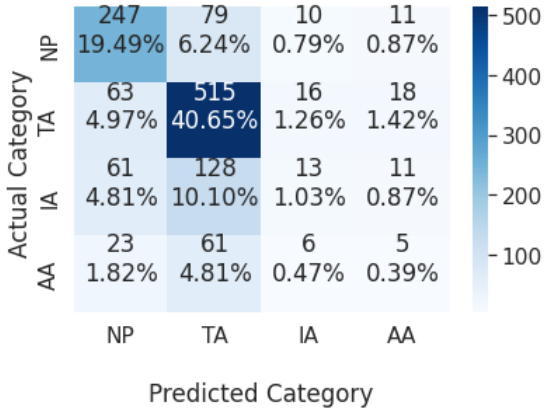

Figura 41 - Matriz de confusão da segunda execução da quarta abordagem

Decidimos avançar com uma nova tentativa deste cenário ainda e obtivemos valores semelhantes aos anteriores, porém um pouco superiores, chegando a um resultado de *validation_accuracy* de 0.6302 na última *epoch* (Figura 42).

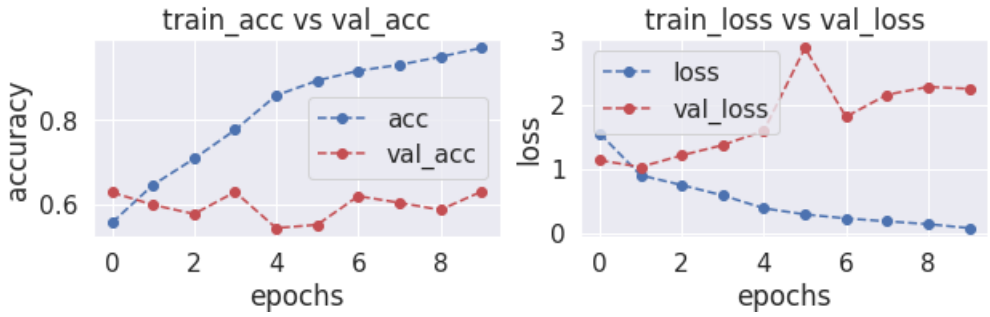

Figura 42 - *Accuracy* e *Loss* ao longo das *epochs* da terceira execução da quarta abordagem

Quanto às *test images* o acerto nas suas *labels* atingiu os 62.59% com a seguinte matriz de correlação (Figura 43):

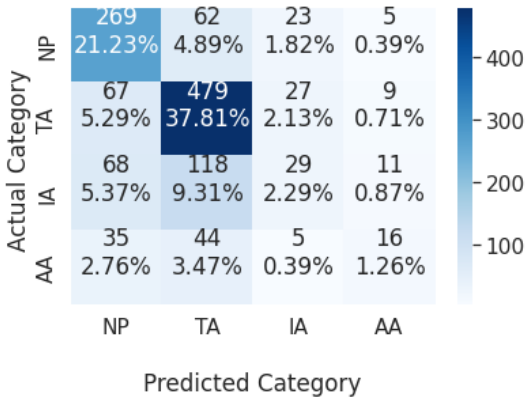

Figura 43 - Matriz de confusão da terceira execução da quarta abordagem

Batch_size 64 e epochs 15

Depois de obter estes resultados além de aumentar as *epochs* na esperança de atingir melhores resultados ainda, ao fim da *epoch* 15 atingimos o valor de 0.6430 (Figura 37 e 38), sendo o mais elevado de todos bem como nas *test images*, acertando em 62.83% das *labels* (Figura 46).

```
Epoch 15/15
64/64 [==============================] - 1010s 16s/step - loss: 0.0670 - accuracy: 0.9761 - val_loss: 3.0201 - val_accuracy: 0.6430
```

Figura 44 - Resultado da *epoch* 15

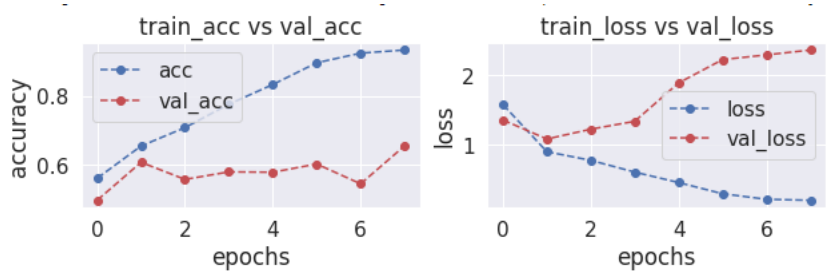

Figura 45 - *Accuracy* e *Loss* ao longo das *epochs* da quarta execução da quarta abordagem

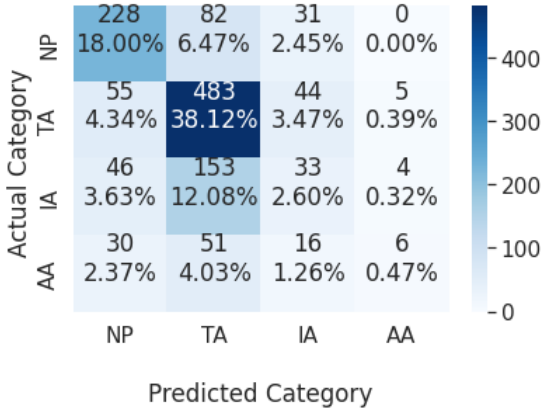

Figura 46 - Matriz de confusão da quarta execução da quarta abordagem

Aplicando o modelo final à aplicação que críamos e usando para efeitos de teste 10 imagens aleatórias de cada uma das classes os resultados que obtivemos foram os seguintes (Figura 47) que resultou numa taxa de acerto de 45% sem cortes e de 47,5% quando há cortes.

Por outro lado, usando uma amostra de 40 imagens de acordo com o balanceamento do *dataset*, ou seja, depois de efetuados os cálculos, resume-se a 3 imagens da categoria AA, 7 imagens da categoria IA, 19 imagens da categoria TA e 11 imagens da categoria NP, obtemos um resultado 67,5% das previsões corretas quando não há corte e 72,5% quando existe corte (Figura 48).

	Sem corte	Com corte
AA1		IA
AA2	NP	NP
AA3	NP	TA
AA4	NP	NP
AA5	NP	NP
AA6	NP	TA
AA7	NP	TA
AA8	NP	TA
AA9	NP	IA
AA10	IA	TA
IA1		TA
IA2	NP	NP
IA3	NP	TA
IA4		TA
IA5	NP	TA
IA6	NP	TA
IA7	NP	TA
IA8	NP	NP
IA9	NP	NP
IA10	NP	TA
TA1	AA	
TA2	IA	
TA3	NP	
TA4	NP	
TA5		IA
TA6		
TA7		
TA8	NP	
TA9		
TA10		
NP1		
NP2		
NP3		
NP4		
NP5		
NP6		
NP7		
NP8		
NP9		
NP10		
certas	0.45	0.475

Figura 47 – Testes na aplicação (não equilibrado)

	Sem corte	Com corte
AA1		IA
AA2	NP	NP
AA3	NP	TA
IA1		TA
IA2	NP	NP
IA3	NP	TA
IA4		TA
IA5	NP	TA
IA6	NP	TA
IA7	NP	TA
TA1	AA	
TA2	IA	
TA3	NP	
TA4	NP	
TA5		IA
TA6		
TA7		
TA8	NP	
TA9		
TA10		
TA11		
TA12		
TA13	IA	
TA14		
TA15		
TA16		
TA17		
TA18		
TA19		
NP1		
NP2		
NP3		
NP4		
NP5		
NP6		
NP7		
NP8		
NP9		
NP10		
NP11		
certas	0.675	0.725

Figura 48 – Testes na aplicação (equilibrado)

Discussão de resultados

Os problemas de classificação são uma das áreas mais pesquisadas no mundo. Os casos de uso estão presentes em quase todos os ambientes de produção e industriais. Reconhecimento de voz, reconhecimento facial, classificação de texto, ou de imagens, como é o caso deste projeto – a lista é interminável.

Os modelos de classificação têm um output discreto, então precisamos de uma métrica que compare classes discretas de alguma forma. As métricas de classificação avaliam o desempenho de um modelo e informam o quão boa ou má é a classificação, mas cada uma delas avalia de forma diferente [16].

A métrica que mais valorizamos para compreender se um modelo é bom ou não é a *accuracy*. A precisão da classificação é talvez a métrica mais simples de usar e implementar e é definida como o número de previsões corretas dividido pelo número total de previsões, multiplicado por 100.

Na primeira abordagem os resultados foram os que mais desiludiram, mas a partir da segunda abordagem foram crescendo mais ou menos ao mesmo ritmo verificando-se melhorias não muito significativas, mas onde conseguimos chegar a um resultado final que ultrapassou a casa dos 60% por diversas vezes e chegou ao seu pico nos 65%.

Ainda relativamente à *accuracy* podemos observar que até à quarta abordagem no início dos treinos rondava os 0.50 e com a substituição do modelo na quarta abordagem facilmente começava já na casa dos 0.60 mostrando então uma melhoria nesse aspeto.

A Matriz de Confusão é uma visualização tabular das *labels* de verdade *versus* as previsões do modelo. Cada coluna da matriz de confusão representa as instâncias numa classe prevista e cada linha representa as instâncias numa classe real. A Matriz de Confusão não é exatamente uma métrica de desempenho, mas uma espécie de base sobre a qual outras métricas avaliam os resultados e dentro dessas outras métricas destacamos a precisão, mas desta vez por cada classe individualmente.

Este outro conceito de precisão surge no contexto de cada classe em que corresponde ao rácio entre o número de verdadeiros positivos e o número total de positivos para essa classe.

Quanto a esta métrica podemos fazer uma observação geral e que se tornou algo evidente e transversal a todas as abordagens que mencionámos. Como se pode observar nas diversas matrizes de confusão há sempre duas classes que surgem com uma cor mais carregada que

significa que a sua precisão é maior, ou seja, são mais vezes classificadas corretamente face às outras duas restantes.

Ao fazer uma análise ao *dataset* este facto é facilmente explicado pelo número de amostras que cada uma destas classes tem, sendo a classe "Típica Aparência" a mais numerosa e igualmente a que melhor precisão tem. De seguida é a "Negativo para a Pneumonia" que aparece com maior número de imagens e também com a segunda melhor precisão no geral.

Um facto curioso a salientar é que o número de imagens da categoria TA face à NP tem uma diferença de 1271 imagens e no entanto facilmente se distinguem das outras duas classes, enquanto a categoria NP tem uma diferença de "apenas" 628 imagens face à IA, quase metade, e, no entanto, a diferença da precisão destas duas últimas é gigante.

Podemos então concluir que provavelmente o modelo tem mais facilidade em aprender características da classe NP do que da classe IA.

Na temática deste projeto surge um conceito bastante relevante e que está relacionado com a *accuracy* e que por vezes acontece chamado de *overfitting*.

O *overfitting* acontece quando um modelo aprende os detalhes e o ruído nos dados de treino na medida em que afeta negativamente o desempenho do modelo em novos dados. Isto significa que o ruído ou as flutuações aleatórias nos dados de treino são captados e aprendidos como conceitos pelo modelo. O problema é que esses conceitos não se aplicam aos novos dados e impactam negativamente a capacidade de generalização dos modelos [17].

Na primeira abordagem claramente conseguimos verificar isto a acontecer uma vez que para os dados de treino o modelo atinge uma precisão (*accuracy*) de 100% e estagna, enquanto que, para novos dados atinge a casa dos 52% e acaba por estagnar igualmente.

Quanto às outras abordagens, principalmente na terceira e quarta, conseguimos também observar isto nos seus gráficos, mas não acontece de forma tão acentuada nem tão explícita.

Outra métrica que também podemos analisar é a *loss* ao longo das *epochs* do treino do modelo que corresponde à penalidade para uma má previsão. Ou seja, a *loss* é um número que indica o quão mau foi a previsão do modelo num único exemplo. Se a previsão do modelo for perfeita, a *loss* é zero; caso contrário, a *loss* é maior. O objetivo de treinar um modelo é encontrar um conjunto de pesos que tenha baixa *loss*, em média, em todos os exemplos [18].

Analisando os resultados obtidos ao longo das diversas abordagens podemos concluir que continua a haver por onde melhorar, tal como na *accuracy*, uma vez que os valores de *training* e os de *validation* durante o treino acabam por se ir distanciando à medida que as

epochs vão avançando, tratando-se então de uma situação de *overfitting* tal como havíamos mencionado anteriormente.

Relativamente aos resultados que obtivemos ao efetuar testes na aplicação conseguimos facilmente concluir que o modelo tem muito mais facilidade em prever a classe "Negativo para a pneumonia" e "Aparência Típica", esta última principalmente quando se efetua um corte na imagem.

Estes resultados vão ao encontro do esperado uma vez que é precisamente nestas duas classes que existem mais imagens durante o treino, sendo o modelo mais eficaz quando se trata de as identificar. Os números finais de 45% e 47,5% também pela mesma razão não se aproximam dos 62,83% de *accuracy* conseguidos nas *test images* da última abordagem uma vez que durante essa previsão não lidámos com um *dataset* equilibrado.

Por outro lado, quando utilizamos o número de imagens para testes correspondente à forma como o *dataset* está distribuído pelas diferentes classes atingimos a taxa de acerto de 67,5% quando não há cortes e de 72,5% quando se efetuam, indo ao encontro dos 62,83% de *accuracy* das *test images* da última abordagem.

Estes resultados vêm então sustentar as conclusões que já tínhamos chegado anteriormente de que o nosso modelo tem mais facilidade em acertar na previsão quando se tratam das classes NP e TA.

Conclusões

O objetivo deste trabalho foi a criação de um algoritmo em *Deep Learning* capaz de detetar a doença COVID-19 através da análise de radiografias de pulmões.

Os resultados obtidos são positivos e à medida que fomos mudando a nossa abordagem a percentagem de acerto nos testes foi crescendo, e assim, ficámos com um algoritmo mais bem treinado e preparado para acertar noutras imagens.

Sendo um trabalho realizado com imagens reais dá-nos mais credibilidade nos resultados, mesmo que o valor da precisão de acerto (*accuracy*) seja de 0.64 e o valor de acerto nos testes seja de 62.83%. Faz, também, com que nos aproximemos e conheçamos mais de uma realidade que cada vez se torna mais visível, como os assistentes virtuais e aplicações capazes de fazer coisas que os humanos não conseguem sem o seu auxílio, como neste caso, detetar se um paciente tem COVID-19 pela análise da radiografia dos seus pulmões.

Como em todos os trabalhos há sempre imprevistos e elementos que correm menos bem, mas são estas situações que nos fazem aprender mais e também os ultrapassámos com sucesso, podemos assim concluir que atingimos os nossos objetivos com sucesso sendo que há sempre margem para melhorar.

Bibliografia ou Referências Bibliográficas

[1] S. C. B. Lo, S. L. A. Lou, J.-S. Lin, M. T. Freedman, M. V. Chien, and S. K. Mun, ‘‘Artificial convolution neural network techniques and applications for lung nodule detection,’’ *IEEE Trans. Med. Imag.*, vol. 14, no. 4, pp. 711–718, Dec. 1995. https://doi.org/10.1109/42.476112

[2] Rajkomar, A., Lingam, S., Taylor, A.G. et al. High-Throughput Classification of Radiographs Using Deep Convolutional Neural Networks. *J Digit Imaging* 30, 95–101 (2017). https://doi.org/10.1007/s10278-016-9914-9

[3] Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., & Rabinovich, A. (2014). Going Deeper with Convolutions (Versão 1). arXiv. https://doi.org/10.48550/ARXIV.1409.4842

[4] Rajpurkar, P., Irvin, J., Zhu, K., Yang, B., Mehta, H., Duan, T., Ding, D., Bagul, A., Langlotz, C., Shpanskaya, K., Lungren, M. P., & Ng, A. Y. (2017). CheXNet: Radiologist-Level Pneumonia Detection on Chest X-Rays with Deep Learning (Versão 3). arXiv. https://doi.org/10.48550/ARXIV.1711.05225

[5] Huang, G., Liu, Z., van der Maaten, L., & Weinberger, K. Q. (2016). Densely Connected Convolutional Networks (Versão 5). arXiv. https://doi.org/10.48550/ARXIV.1608.06993

[6] Shen, W., Zhou, M., Yang, F., Yang, C., Tian, J. (2015). Multi-scale Convolutional Neural Networks for Lung Nodule Classification. In: Ourselin, S., Alexander, D., Westin, CF., Cardoso, M. (eds) Information Processing in Medical Imaging. IPMI 2015. Lecture Notes in Computer Science(), vol 9123. Springer, Cham. https://doi.org/10.1007/978-3-319-19992-4_46

[7] Ker, J., Wang, L., Rao, J., & Lim, T. (2018). Deep Learning Applications in Medical Image Analysis. Em IEEE Access (Vol. 6, pp. 9375–9389). Institute of Electrical and Electronics Engineers (IEEE). https://doi.org/10.1109/access.2017.2788044

[8] Causey, J.L., Zhang, J., Ma, S. et al. Highly accurate model for prediction of lung nodule malignancy with CT scans. Sci Rep 8, 9286 (2018). https://doi.org/10.1038/s41598-018-27569-w

[9] Yaniv Bar, Idit Diamant, Lior Wolf, Hayit Greenspan, "Deep learning with non-medical training used for chest pathology identification," Proc. SPIE 9414, Medical Imaging 2015: Computer-Aided Diagnosis, 94140V (20 March 2015); https://doi.org/10.1117/12.2083124

[10] IBM Cloud Education. (2020, 17 de Agosto). *Types of neural networks*. IBM. https://www.ibm.com/cloud/learn/neural-networks - toc-types-of-n-YgdI1-Kt

[11] AI Wiki. (s.d.). *Weights and Biases*. https://machine-learning.paperspace.com/wiki/weights-and-biases

[12] IBM Cloud Education. (2020, 1 de Maio). *Deep Learning*. IBM. https://www.ibm.com/cloud/learn/deep-learning

[13] Sharma, S., Sharma S. & Athaiya A. (2020). Activation functions in neural networks. *International Journal of Engineering Applied Sciences and Technology*, *4*(12), 310-316.

http://www.ijeast.com/papers/310-316,Tesma412,IJEAST.pdf

[14] Beacker, D. (2018, 7 de Maio). *Rectified Linear Units (ReLU) in Deep Learning*. Kaggle. https://www.kaggle.com/code/dansbecker/rectified-linear-units-relu-in-deep-learning/notebook

[15] Brownlee, J. (2019, 19 de Abril). *A Gentle Introduction to Padding and Stride for Convolutional Neural Networks*. Machine Learning Mastery. https://machinelearningmastery.com/padding-and-stride-for-convolutional-neural-networks/

[16] Bajaj, A. (2022, 21 de Juho). Performance Metrics in Machine Learning. *Neptune Blog*. https://neptune.ai/blog/performance-metrics-in-machine-learning-complete-guide

[17] Brownlee, J. (2019, 12 de Agosto). *Overfitting and Underfitting With Machine Learning Algorithms*. Machine Learning Mastery. https://machinelearningmastery.com/overfitting-and-underfitting-with-machine-learning-algorithms/

[18] Google Machine Learning Education. (2022, 18 de Julho). *Descending into ML: Training and Loss*. Google Developers. https://developers.google.com/machine-learning/crash-course/descending-into-ml/training-and-loss

[19] Moroney, L. [lmoroney]. & Tensorflow. (2019, 11 de Setembro). *Introducing convolutional neural networks (ML Zero to Hero - Part 3)*. [Vídeo]. Youtube. https://www.youtube.com/watch?v=x_VrgWTKkiM

[20] Tan, M. (2019, 29 de Maio). EfficientNet: Improving Accuracy and Efficiency through AutoML and Model Scaling. *Google AI Blog*. https://ai.googleblog.com/2019/05/efficientnet-improving-accuracy-and.html~

[21] Sarkar, A. (2021, 8 de Maio). Understanding EfficientNet — The most powerful CNN architecture. *Medium*. https://medium.com/mlearning-ai/understanding-efficientnet-the-most-powerful-cnn-architecture-eaeb40386fad

[22] Society for Imaging Informatics in Medicine (2021, 10 de Agosto). *SIIM-FISABIO-RSNA COVID-19 Detection – Description*. Kaggle. https://www.kaggle.com/competitions/siim-covid19-detection

[23] Viceri. (2020, 29 de Julho). *Arquiteturas de Redes Neurais Convolucionais para reconhecimento de imagens*. https://viceri.com.br/insights/arquiteturas-de-redes-neurais-convolucionais-para-reconhecimento-de-imagens/

[24] Keras. https://keras.io/

[25] He, K., Zhang, X., Ren, S., & Sun, J. (2015). Deep Residual Learning for Image Recognition. https://doi.org/10.48550/arXiv.1512.03385

[26] Tan, M., & Le, Q. v. (2019). EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks. https://doi.org/10.48550/arXiv.1905.11946

[27] Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., & Chen, L.-C. (2018). MobileNetV2: Inverted Residuals and Linear Bottlenecks. https://doi.org/10.48550/arXiv.1801.04381